INF523: Computer Systems Assurance

Measuring Security

http://ccss.usc.edu/523

Prof. Clifford Neuman

Lecture 2
4 September 2020
Online

Course Identification

- DSci 523
  - Assurance in Cyberspace
  - 4.0 units
- Class meeting schedule
  - 1:00-4:20pm Friday
  - Room: Online
  - Class communication
  - For now, bcn@isi.edu, include DSci523: in subject

General Course Information

- Professor office hours
  - Tuesday 2:30PM to 4:00PM
  - Other times by appointment
    Zoom link was already sent to students
  - E-mail: bcn@isi.edu

- TA/Grader for the class
  - TBA

Reading for This Week

- TCSEC, pp. 10, 50-53, 62-63, 67-68, 77-79
- Common Criteria, Part 3, pp. 15-17, 44-45
- Final Evaluation Report, Gemini Trusted Network Processor – Section 7 (GTNP-NCSC-FER-94-008.pdf)
  - [All above on DEN in "Readings" module]
- SSE-CMM/ISO 21827 Capability Maturity Model (http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html - search for "ISO 21827")
- Build Security In Maturity Model (BSIMM) (http://www.bsimm.com/)
- Microsoft Security Development Lifecycle (http://www.microsoft.com/en-us/download/details.aspx?id=29884)

Case Studies Phase 1

- Proposal Assignment for Case Study:
  - Case studies presented in two stages.
    - A 10 minute presentation on the 25th of September
      - The presentation should identify a class of systems or a particular system or product (e.g. The could, ApplePay, the i-phone, Android, SE-Linux, etc)
        - » Send me email by 9/5 for approval of topic
      - Explain (though not necessarily answer) the assurance issues that need to be met by the identified system.
      - Identify the consequences of security failure in such systems.
      - Discuss where one will look to answer those questions.
    - Topics will be grouped into related systems

Case Study Phase 2

- Students in groups will prepare a case study presentation of approximately 30 minutes per student.
  - Each student will take a section
  - Questions to be addressed include:
    - Where is the TCB(s) in the system(s)
      - How does the system utilize minimization
    - Where should the TCB(s) be.
    - What assurance techniques are applied
    - How to improve the security of the system
- These will be presented near the end of the semester.

Example Case Studies

- You can find examples of case studies by students from previous years (posted with their permission):
  - System Security Enhancements in Windows 10, Summer 2017 Case Study, Samuel Aspiranti
  - Security Enhanced LINUX, Fall 2015 Case Study - John Bush and Heather Romero

INF523: Computer Systems Assurance

Measuring Security (continued)

http://ccss.usc.edu/523

Prof. Clifford Neuman

Lecture 2
4 September 2020
Online

Foundations of Computer Security

1. A security policy that states the laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information
2. The functionality of internal mechanisms to enforce that security policy
3. Assurance that the mechanisms correctly enforce the security policy

Review of Key Concepts

- A system *security policy* is a statement of requirements that defines security expectations
- *Cyber assurance* is trust that system correctly enforces security policy
- *Trust* must be gained through evidence, using assurance techniques
- *Assurance techniques* must be applied at all stages of the system lifecycle

Assurance in the System Lifecycle

- The system's security policy is internally consistent and reflects the requirements of the organization
- There are sufficient security functions to support the security policy
- The security functions meet a desired set of properties (and only those properties)
- The functions are implemented correctly
- The assurances hold up through the manufacturing, delivery, and other stages of the system lifecycle

High vs. Low Assurance

- We provide assurance through techniques such as structured design processes, documentation, and testing
- Higher assurance through use of more, and more rigorous, processes, documentation, and testing
  - Intuitively, compare a little testing vs. a great deal of testing
- There are some fundamental ways to organize these processes to make the job easier and to have a more robust product

A Good Start

- Implement a reference monitor
  - Tamperproof
  - Always invoked
  - "**Small enough to be subject to independent testing, the completeness of which can be assured**" (TCSEC)
- *Trusted Computing Base (TCB)*
  - Combination of hardware, software, and firmware that is responsible for enforcing the system's security policy
- **Minimization of the complexity** in the TCB is a goal
- That puts more code outside the TCB, but so what?
  - **We are concerned here with the security policy only**

Question

- A vendor says they have thoroughly tested their system and found no flaws. They say their system can be “trusted” with sensitive information. Would you believe it?
  - Why or why not?

Question

- If Microsoft tomorrow was to announce that "we've identified the security flaws in Windows 10 and came up with new security requirements that we implemented in Windows 11, so Windows 11 is totally secure", would you believe it?
  - Why or why not?
  - If not, what would it take for you to believe it?

Question

- Your company wants to build a “secure” application that will run on Windows (version 10 and up)
  - You can expend as much effort, tools, and other resources necessary to make the application secure
  - You determine the security req’s, model threats, create a policy, generate the security req’s and use formal methods to ensure the security features map to the policy, create a design and use formal methods to map the design to the security req’s, implement the design and formally “prove” that the code satisfies the specification.
- Is your application high-assurance?

What do these examples tell us?

- Testing cannot (in other than trivially reduced circumstances) ever be complete
- Vendors' claims should always be suspect until sufficient evidence is provided (if it exists)
- You can't spin gold out of straw
  - Must consider entire TCB, not just components
- *Assurance argument* based on totality of evidence for all stages of the system lifecycle
  - Gaps in the assurance argument are not good
  - Inconsistencies in the assurance argument are not good

Evaluating Assurance Arguments

- How does a normal purchaser of software and systems evaluate vendor security claims and evidence?

- Depend on 3rd-party expert evaluators
  - Presumed: Trained, experienced, good judgment, unbiased

Topics Covered in this Lecture

- **Assurance Requirements in Evaluation Criteria**
  - Assurance requirements at different evaluation levels
- **Capability Maturity Models**
  - An approach for assessing the capability of a vendor to produce a secure system
- **Microsoft's SDLC**
  - A contemporary, real-world assurance process at a major software company

An Example Evaluation Criteria

- The Orange Book View of Assurance
  - Orange Book = TCSEC (http://csrc.nist.gov/publications/history/dod85.pdf)
- No longer used, but good example of what is needed to make assurance arguments

Orange Book Assurance

- The Assurance Control Objective
  - Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy and must not distort the intent of that policy. **Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle.**
    - Operational Assurance
    - System Architecture
    - System Integrity
    - Covert Channel Analysis
    - Trusted Facility Management
    - Trusted Recovery

A Complication

- Should we always expect very high assurance evidence in every instance of security solutions / technology?
  - What is the case for? What is the case against?
- Consider the padlock industry
  - There are many different locks on the market
  - There is the notion of best, better, good, good enough and sort of ok
  - We make a risk judgment every time we buy one
  - Implicit assurance argument about sufficiency to mitigate the threat and “fit for purpose” (trustworthiness)
  - But the assurance is not the same across the lock space

Subtleties – Balanced Assurance

- Suppose you have a really strong, high assurance perimeter control system. You've made a heavy investment to make sure only permitted individuals are able to have access through this perimeter. By allowing those individuals through the perimeter you have acknowledged great trust. Each of them has an office inside the perimeter. each office has a lock on it. What threat does the office lock mitigate?
- Does the lock on the office need to be a super strong high assurance lock, comparable with the perimeter?

Subtleties - Composability

- Consider a high security jail that is built by assembling modules from various different (likely by a lowest bid, but no kickback suppliers)
- There are modules for the walls (with and without windows) doors, cells, services (food, medical and other). The modules "snap" together into the jail.
- You, of course, are worried about the management of this jail and very worried that someone will try to break out of this jail.
  - Each module is constructed to your specification with your assurance requirements and validated at the manufacturer
  - What would be the assurance argument for the jail as a unit?
  - How would you go at this problem
  - How would you decide if / when the jail was fit for purpose?

Subtleties – Assurance and "the right stuff"

- Vendors market what sells, independently of what matters
- Vendors might try to pad the assurance with stuff that is not overly germane to your specific needs
  - For example paint quality on patrol cars, The vendor could make a big deal out of this but that is not one of your most important concern
- Consider intrusion detection systems. You want an IDS that stops all intrusions. What would be an assurance argument that the Vendor might present to you to substantiate that claim and what might be the important part(s) he/she might leave out?

TCSEC Classes

- D – Minimal Protection
- C – Discretionary Protection
  - C1 – Discretionary Security Protection - DAC
  - C2 – Controlled Access Protection – DAC + audit, etc.
- B – Mandatory Protection
  - B1 – Labeled Security Protection (has MAC labels)
  - B2 – Structured Protection (FSPM)
  - B3 – Security Domains (implements RM)
- A – Verified Protection
  - A1 – Verified Design (formal design, spec, and verify)

TCSEC summary
Security Policy

Figure 1.: Summary of the TCSEC

SECURITY POLICY	C1	C2	B1	B2	B3	A1
Discretionary Access Control						
Object Reuse						
Labels						
Label Integrity						
Exportation of labelled information						
Exportation to Multi-Level Devices						
Exportation to Single-Level Devices						
Labelling Human-Readable Output						
Mandatory Access Control						
Subject Sensitivity Labels						
Device Labels						

LEGEND

No additional requirements	
New or enhanced requirements	
No requirements	

TCSEC summary Accountability

Figure 1.: Summary of the TCSEC

	C1	C2	B1	B2	B3	A1
ACCOUNTABILITY						
Identification and Authentication						
Audit						
Trusted Path						

LEGEND

No additional requirements	
New or enhanced requirements	
No requirements	

TCSEC summary Documentation

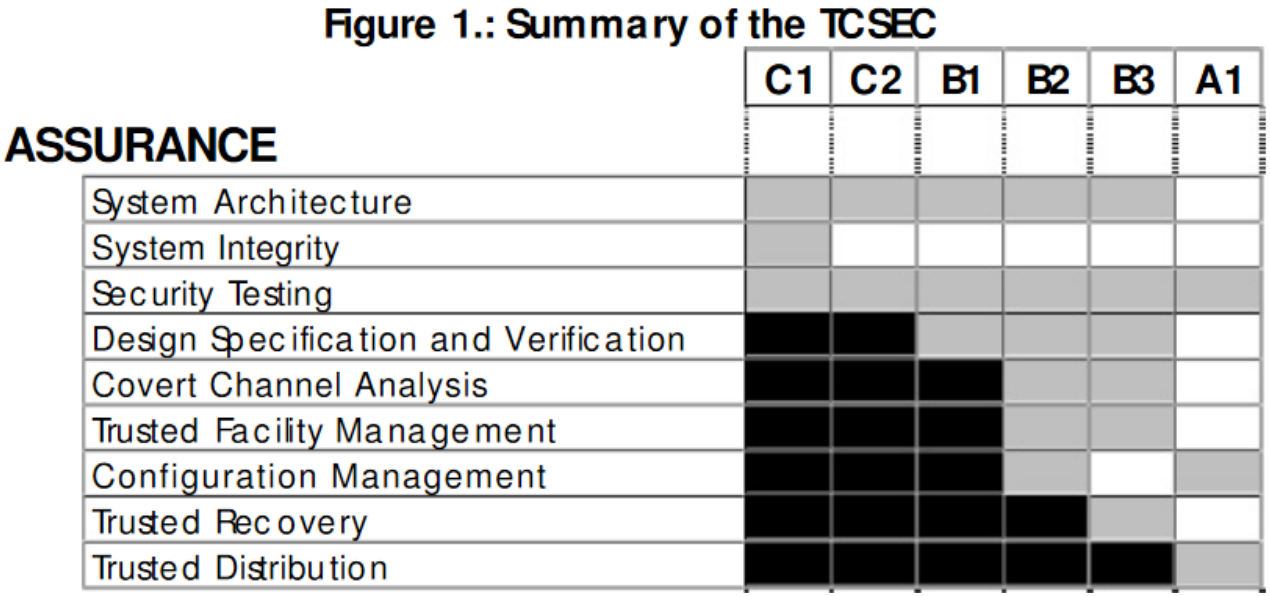
Figure 1.: Summary of the TCSEC

	C1	C2	B1	B2	B3	A1
DOCUMENTATION						
Security Features User's Guide						
Trusted Facility Manual						
Test Documentation						
Design Documentation						

LEGEND

No additional requirements	
New or enhanced requirements	
No requirements	

TCSEC summary Assurance

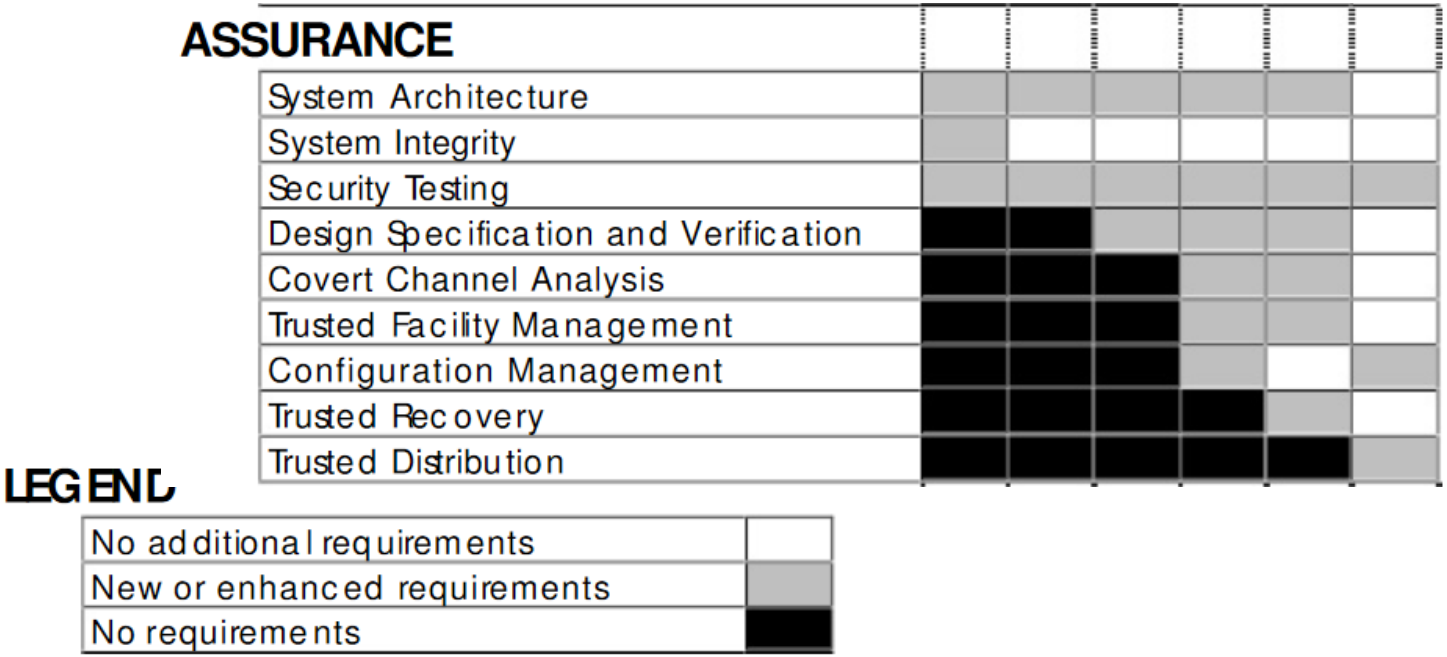


Orange Book Assurance Requirements

- Orange book has different assurance requirements for different classes
- Except for system integrity, each of the assurance measures is graded
  - The measure is incrementally increased as the threat mitigation expectation of the system increases
- Logical grouping of the assurance measures.
  - Design Specification and Verification is not part of the DAC-only policy systems (Why?)
  - In fact, most of the assurance measures only apply to systems that provide MAC policy enforcement
  - I.e., they apply to the machine in the middle (the RM)

Example Local Grouping

- A1's chief concern is subversion, so yet more requirements for, e.g., trusted distribution



Example of Graded Assurance Measures

- We'll look at Security Testing assurance requirements
- The security testing "ramp"
  - Higher assurance for higher security class

Class C1 Security Testing

- **The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB. (See the Security Testing Guidelines.)**

Class C2 Security Testing

- The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB. **Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data. (See the Security Testing guidelines**.)

Class B1 Security Testing

- The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. **A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced. (See the Security Testing Guidelines.)**

Class B2 Security Testing

- The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. **The TCB shall be found resistant to penetration**. All discovered flaws shall be corrected and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced. **Testing shall demonstrate that the TCB implementation is consistent with the formal top level specification.** (See the Security Testing Guidelines.)

Class B3 Security Testing

- The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. The TCB shall be found resistant to penetration. All discovered flaws shall be corrected and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced. Testing shall demonstrate that the TCB implementation is consistent with the formal top level specification. (See the Security Testing Guidelines.) **No design flaws and no more than a few correctable implementation flaws may be found during testing and there shall be reasonable confidence that few remain.**

Class A1 Security Testing

- The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. The TCB shall be found resistant to penetration. All discovered flaws shall be corrected and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced. Testing shall demonstrate that the TCB implementation is consistent with the formal top level specification. (See the Security Testing Guidelines.) No design flaws and no more than a few correctable implementation flaws may be found during testing and there shall be reasonable confidence that few remain. **Manual or other mapping of the FTLS to the source code may form a basis for penetration testing.**

Security Testing Guidelines – Division C

- At least two people with Bachelor degrees in CS
- Must be familiar with “flaw hypothesis” testing methodology (a form of pen-testing)
  - Create list of possible flaws through analysis of specs and documentation of a system
  - Prioritize based on likelihood and ease of exploit
- Must carry out system developer-defined tests
- Must independently design and implement at least 5 tests
  - 1 month <= testing <= 3 months
  - 20 hours <= testing for each team member

Security Testing Guidelines – Division B

- At least two people with Bachelor degrees in CS
- At least one person with Master's Degree in CS
- Must be fluent in TCB implementation language(s)
- Must be experienced with assembly language
- Must have completed system developer's internals course for the system
- At least one team member must have completed a security test on another system
- Team must independently design and implement at least 15 tests
  - 2 months <= testing <= 4 months
  - 30 hours <= testing for each team member

Security Testing Guidelines – Division A

- At least one person with Bachelor degree in CS
- At least two people with Master's degrees in CS
- At least one team member must be familiar enough with the system hardware to understand diagnostic programs and HW documentation
- At least 2 team members must have completed a security test on another system
- At least one team member must have demonstrated expertise on the system under test sufficient to, e.g., add a device driver
- Team must independently design and implement at least 25 tests
  - 3 months <= testing <= 6 months
  - 50 hours <= testing for each team member

Example FER for Class A1 System

- Final Evaluation Report (FER) for Gemini Trusted Network Processor
- The FER provides descriptions of the following system aspects:
  - Architecture HW (e.g., segmentation and rings) and SW
  - Design
  - Mapping to policy model (BLP)
  - Assurance
    - Architecture – modularization and layering, data hiding, minimization
    - System Integrity
    - Covert channel analysis and testing
    - Trusted Recovery
    - Security Testing – functional and exception testing
    - Design specification and verification
    - Configuration management and maintenance
    - Trusted Distribution

Assurance Takeaway from the TCSEC

- Tied to security policy and FSPM
- Based on notion of TCB
  - All the HW/SW/FW in the system that enforces the policy
  - The other stuff doesn't matter from security pov
- Identifies SDLC + assurance techniques for each step
- Trades off level of assurance against required protection
- Permits different levels of effort against different layers of the system ("balanced assurance")
- Considers composition of parts (TNI)

Topics Covered in this Lecture

- **Assurance Requirements in Evaluation Criteria**
  - Assurance requirements at different evaluation levels
- **Capability Maturity Models**
  - An approach for assessing the capability of a vendor to produce a secure system
- **Microsoft's SDLC**
  - A contemporary, real-world assurance process at a major software company

Capability Maturity Models

- How can a user assess the security of a product?
  - After lengthy, third-party evaluation (but product may be nearly obsolete by then)
  - Immediately, but assurance rests on claims by vendor
- Improve assurance and time-to-market by *pre-reviewing security engineering processes of vendor*
  - Third-party review of vendor security engineering processes (capabilities)
  - Focus on measuring organization competency (maturity) and improvements

Capability Maturity Models (2)

- Goals:
  - Continuity - knowledge acquired in previous efforts is used in future efforts
  - Repeatability - a way to ensure that projects can repeat a successful effort
  - Efficiency - a way to help both developers and evaluators work more efficiently
  - Assurance - confidence that security needs are being addressed.

Capability Maturity Models (3)

- Focus on existence of process
- Organizations appraised by third-party
- Score based on number and sophistication of practices followed
- Goal for vendor is to be appraised high for competitive advantage
- Acquirers can put required CMM level in RFPs
- Similar to “six sigma” or ISO 9000 certification for quality and process improvement

System Security Engineering - Capability Maturity Model (SSE-CMM, ISO/IEC 21827)

- Covers entire organization, including management as well as engineering
- Based on observed engineering best practices at over 50 large organizations (including multi-nationals)
- Addresses the complete product life cycle:
  - Concept definition
  - Development
  - Production
  - Utilization
  - Support
  - Retirement

System Security Engineering

Capability Maturity Model (SSE-CMM, ISO/IEC 21827)

- Two dimensions: *domain* and *capability*
- Domain is "base practices" of security engineering
  - E.g., Base Practice 05.02, "Identify System Security Vulnerabilities"
- Capability is "generic practices" that should be part of base practices
  - E.g., Generic Practice 2.1.1, "Allocate Resources"
- Intersection indicates an organization's capability to perform a particular activity
  - E.g., "Does the organization allocate resources for use in identifying system security vulnerabilities?"

Base Practices

- Apply to entire life cycle
- Represents a “best practice” in the security community
- Organized into Process Areas
  - Not all organizations have same needs or goals
  - Some provide products, some systems, others services

Process Areas for Systems Security Engineering

- PA01 Administer Security Controls
- PA02 Assess Impact
- PA03 Assess Security Risk
- PA04 Assess Threat
- PA05 Assess Vulnerability
- PA06 Build Assurance Argument
- PA07 Coordinate Security
- PA08 Monitor Security Posture
- PA09 Provide Security Input
- PA10 Specify Security Needs
- PA11 Verify and Validate Security
- Additional process areas for project and org. practices

PA05 - Assess Vulnerability

- Description:
  - Identify and characterize security vulnerabilities
    - Analyze system assets
    - Define specific vulnerabilities
    - Provide an assessment of the overall system vulnerability
  - Performed any time during a system's life-cycle
- Goals:
  - An understanding of system security vulnerabilities within a defined environment is achieved

PA05 - Assess Vulnerability

- Base Practice List:
  - BP.05.01 Select the methods, techniques, and criteria by which security system vulnerabilities in a defined environment are identified and characterized
  - BP.05.02 Identify system security vulnerabilities
  - BP.05.03 Gather data related to the properties of the vulnerabilities
  - BP.05.04 Assess the system vulnerability and aggregate vulnerabilities that result from specific vulnerabilities and combinations of specific vulnerabilities
  - BP.05.05 Monitor ongoing changes in the applicable vulnerabilities and changes to their characteristics

BP.05.02 – Identify Vulnerabilities

- Description: The methodology of attack scenarios (description of specific attacks) as developed in BP.05.01 should be followed to the extent that vulnerabilities are validated. All system vulnerabilities discovered should be recorded.
- Example Work Products:
  - Vulnerability list describes the vulnerability of the system to various attacks
  - Penetration profile includes results of the attack testing (e.g., vulnerabilities)

SSE-CMM Capability Levels

0 INITIAL

1 PERFORMED INFORMALLY

- **Base practices performed**

2 PLANNED & TRACKED

- **Planning performance**
- **Disciplined performance**
- **Verifying performance**
- **Tracking performance**

3 WELL-DEFINED

- **Defining a standard process**
- **Perform the defined process**
- **Coordinate practices**

4 QUANTITATIVELY CONTROLLED

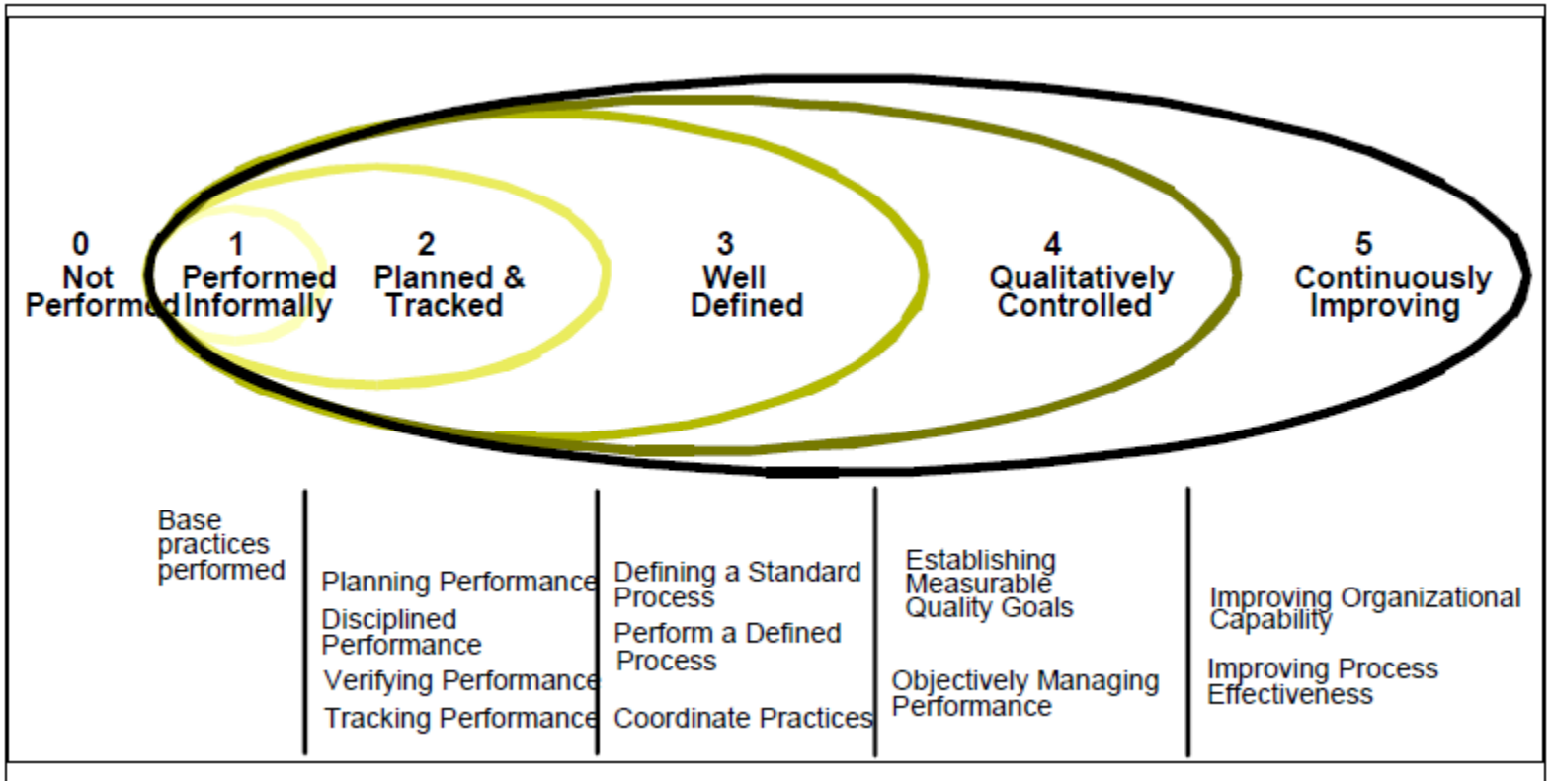
- **Establishing measurable quality goals**
- **Objectively managing performance**

5 CONTINUOUSLY IMPROVING

- **Improving organizational capability**
- **Improving process effectiveness**

Capability Levels

- Capability levels represent the maturity of orgs



Generic Practices

- Organized by capability level
  - I.e., add additional generic practices at each level
- Each level decomposed into set of common features
- Each set of common features consists of set of generic practices

Capability Level 1

- Common Feature 1.1 - Base Practices Are Performed
- GP 1.1.1 - Perform the Process

Capability Level 2

- Common Features:
  - Common Feature 2.1 - Planning Performance
  - Common Feature 2.2 - Disciplined Performance
  - Common Feature 2.3 - Verifying Performance
  - Common Feature 2.4 - Tracking Performance

Capability Level 2 Common Feature 1

- Focuses on aspects of planning to perform the Process Area and its associated Base Practices
- Generic Practices:
  - GP 2.1.1 - Allocate Resources
  - GP 2.1.2 - Assign Responsibilities
  - GP 2.1.3 - Document the Process
  - GP 2.1.4 - Provide Tools
  - GP 2.1.5 - Ensure Training
  - GP 2.1.6 - Plan the Process

Capability Level 3

- Common Feature 3.1 - Defining a Standard Process
  - GP 3.1.1 - Standardize the Process
  - GP 3.1.2 - Tailor the Standard Process
- Common Feature 3.2 - Perform the Defined Process
  - GP 3.2.1 - Use a Well-Defined Process
  - GP 3.2.2 - Perform Defect Reviews
  - GP 3.2.3 Use Well-Defined Data
- Common Feature 3.3 - Coordinate Practices
  - GP 3.3.1 - Perform Intra-Group Coordination
  - GP 3.3.2 - Perform Inter-Group Coordination
  - GP 3.3.3 Perform External Coordination

Capability Level 4

- Common Feature 4.1 - Establishing Measurable Quality Goals
  - GP 4.1.1 - Establish Quality Goals
- Common Feature 4.2 - Objectively Managing Performance
  - GP 4.2.1 - Determine Process Capability
  - GP 4.2.2 - Use Process Capability

Capability Level 5

- Common Feature 5.1 - Improving Organizational Capability
  - GP 5.1.1 - Establish Process Effectiveness Goals
  - GP 5.1.2 - Continuously Improve the Standard Process
- Common Feature 5.2 - Improving Process Effectiveness
  - GP 5.2.1 - Perform Causal Analysis
  - GP 5.2.2 - Eliminate Defect Causes
  - GP 5.2.3 - Continuously Improve the Defined Process

Limits of Capability Maturity Models

- Doesn't consider security policy (e.g., no MAC)
- Measures existence of process, but not quality
  - Existence is (roughly) objective – measurement is possible
  - Quality is subjective – no standard for comparison
- Doesn't guarantee good results
  - Does not measure effectiveness of processes
  - Can do everything but achieve nothing
- Non-uniformity of appraisals
- Misunderstanding of model and its use
  - Doesn't replace testing/evaluation
- Doesn't consider subversion

Build Security In Maturity Model (BSIMM)

- On-going project (http://www.bsimm.com/)
- Collected practices observed at real-world places ("67 leading software security initiatives" at leading companies)
- Compare target against comparison set
  - "Here's what everybody else is doing"

Adobe
Aetna
Bank of America
Box
Capital One
Citi
Comerica Bank
EMC
Epsilon
F-Secure
Fannie Mae
Fidelity
Goldman Sachs
HSBC
Intel
Intuit
JPMorgan Chase & Co.
Lender Processing Services Inc.
Marks and Spencer
Mashery
McAfee
McKesson
Microsoft
NetSuite
Neustar
Nokia
Nokia Siemens Networks
PayPal
Pearson Learning Technologies
QUALCOMM
Rackspace
Salesforce
Sallie Mae
SAP
Sony Mobile
Standard Life
SWIFT
Symantec
Telecom Italia
Thomson Reuters
TomTom
T. Rowe Price
Vanguard
Visa
VMware
Wells Fargo
Zynga

BSIMM Domains

- *Governance*: organization, management, and measurement of a software security initiative
- *Intelligence*: Collection of “corporate security knowledge”
- *SDL Touchpoints*: analysis and assurance of software development artifacts and processes
- *Deployment*: software configuration, maintenance, and other environment issues that have direct impact on software security

BSIMM Practices

- 112 activities organized into 12 practices in 4 domains
  - Activities organized into increasing level of

Governance	Intelligence	SDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

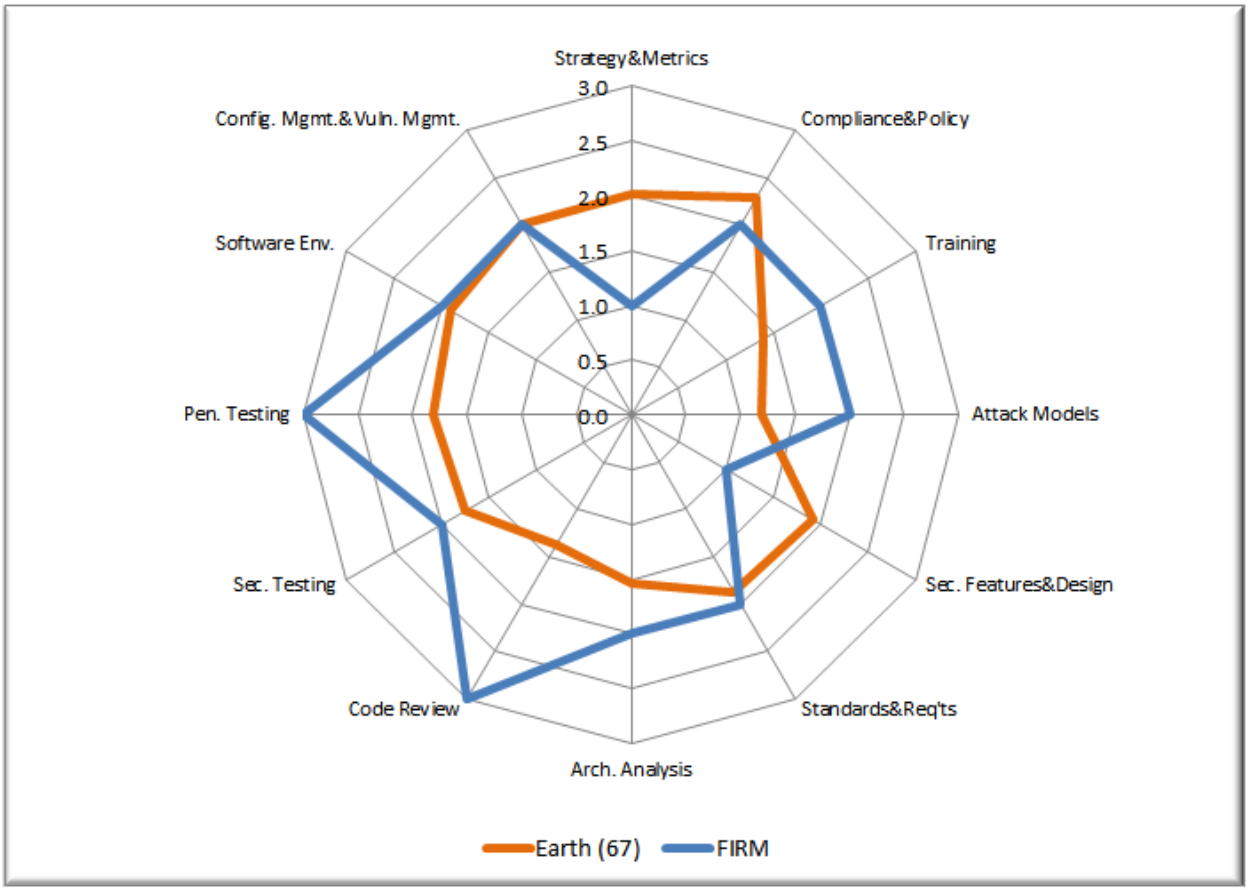
Example Practice: Architecture Analysis

Capturing software architecture diagrams, applying lists of risks and threats, adopting a process for review, building an assessment and remediation plan.			
Activity ID	Objective	Activity	Level
AA1.1	get started with AA	perform security feature review	1
AA1.2	demonstrate value of AA with real data	perform design review for high-risk applications	
AA1.3	build internal capability on security architecture	have SSG lead review efforts	
AA1.4	have a lightweight approach to risk classification and prioritization	use a risk questionnaire to rank applications	
AA2.1	model objects	define and use AA process	2
AA2.2	promote a common language for describing architecture	standardize architectural descriptions (including data flow)	
AA2.3	build capability organization-wide	make SSG available as AA resource or mentor	
AA3.1	build capabilities organization-wide	have software architects lead review efforts	3
AA3.2	build proactive security architecture	drive analysis results into standard architectural patterns	

Example Activity: AA1.1

- **Perform security feature review**
  - Identify security features in an application (authentication, access control, use of cryptography, etc.)
  - Look for problems that would cause these features to fail at their purpose or otherwise prove insufficient
- Example: “a system that was subject to escalation of privilege attacks because of broken access control would both be identified in this kind of review.”
- Where is the policy mentioned?

BSIMM Activity Coverage Graph



Source: http://www.bsimm.com/community/

Limits of BSIMM

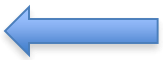
- Divorced from consideration of security policy
- Ad hoc. “Common practices” are not necessarily complete or even good
- Notes existence of process, but not quality
- No coding standards, except for reusing “mature” and “secure-by-design” frameworks
- Doesn’t consider subversion
  - Version management?

Microsoft Security Development Lifecycle (SDL)

- Mandated for use at Microsoft since 2004
- 7 phases:
  - Training
  - Requirements
  - Design
  - Implementation
  - Verification
  - Release
  - Response

Training Phase

- For all developers
- Major topics:
  - Common excuses for not fixing bugs
  - Secure design - identify and avoid issues that can lead to compromise (does this imply there is a policy?)
  - Threat modeling
  - Secure coding
  - Security testing
  - Privacy best practices



At last, a policy?

Requirements Phase

- Establish security and privacy requirements
  - Based on what criteria?
- Create quality gates/bug bars
  - E.g., require fixing of all “critical” vulnerabilities before release
- Perform security and privacy risk assessments
  - Determine need for threat modeling and security design reviews of components
  - Based on costs and regulatory requirements

Design Phase

- Establish design requirements
  - Validate all design specifications against a functional specification
  - Presumably, security functions identified in previous phase
- Perform attack surface analysis and reduction
  - Disable or restrict access to services
  - Reduce privilege
  - Layered defenses
- Use threat modeling
  - Microsoft STRIDE approach
  - We'll look at that in a future lecture

Implementation Phase

- Use approved tools
  - Approved compilers and linkers (and associated options and warnings)
- Deprecate unsafe functions and APIs
  - Ban unsafe functions
    - e.g., *gets* does not check bounds; use *fgets* instead
  - Use newer header files, compilers, or code scanning tools
- Perform static analysis
  - Automated tool scans code for common flaws without executing the code
  - We'll look at this in a future lecture

Verification Phase

- Perform Dynamic Analysis
  - Run-time verification of software functionality
  - Checks for memory corruption, user privilege issues, etc.
  - We'll discuss this topic in a future lecture
- Perform Fuzz Testing
  - Try to induce failure through malformed or random input data
- Conduct attack surface review
  - As in design phase, but on system basis
  - Tool takes snapshot of Windows system state before and after installation of product (http://www.microsoft.com/en-us/download/details.aspx?id=24487)

Attack Surface

- *Attack surface*: Exposed parts of systems that may have exploitable vulnerabilities, e.g.,
  - Open ports on outward facing web and other servers
  - Code that processes incoming data
  - Employees who can be “socially-engineered”
- Reference monitor (RM) abstraction helps
  - TCB shrinks attack surface
  - But still must consider threats to RM
- May have low-assurance components, not RM
  - E.g., your typical corporate IT network
  - Can still try to show non-bypassable, tamper-proof, and make assurance arguments
  - Security mechanisms probably have vulnerabilities!

Release Phase

- Create an Incident Response Plan
  - Includes emergency contacts and maintenance plan
  - For incidents with both internally developed and licensed software
- Conduct final security review
  - Examine artifacts of security activities (e.g., threat models) against quality gates/bug bars
- Certify release and archive
  - Attest that all security and privacy requirements met
  - Archive all specs, source, binaries, SDL artifacts, etc.

Limits of Microsoft SDL

- Still little or no connection to policy
  - Policy is implicit, perhaps
  - How do you know you have identified all (well, most) threats and requirements?
- Does it work? (http://www.gfi.com/blog/report-most-vulnerable-operating-systems-and-applications-in-2013/)
  - Application in 2013 with the most critical flaws discovered: Microsoft Internet Explorer
  - OS in 2013 with the most critical flaws discovered: Microsoft Windows Server 2008
    - Next 5: Windows 7, Vista, XP, Windows Server 2003, Windows 8

Why Is Microsoft Software Still so Vulnerable?

- Different targets
  - TCSEC: TCB that enforces a policy
  - MSDL: Any software; no policy
- Different requirements
  - TCSEC: Only goal is enforcing security policy
  - MSDL: “Security and privacy” requirements possibly secondary to other requirements
    - E.g., legacy code components and support
- Different testing
  - TCSEC: Think like an attacker – directed and prioritized
  - MSDL: Passive search for flaws – random, flat

Some Other Models

- SAFECode "Fundamental Practices for Secure Software Development"
  - http://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf
  - Aimed at reducing software weaknesses
- OWASP Software Assurance Maturity Model (OpenSAMM)
  https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model
  - Many similarities to BSIMM – e.g., 4 domains, 12 practices
  - But more prescriptive focus
- Dept. of Homeland Security "Build Security In" initiative https://buildsecurityin.us-cert.gov/

NIST 800-171

- Some assessment methodologies are checklists of best practices that an organization should follow.
  - NIST SP 800-171: Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations
    - Lists 110 requirements in 14 areas (sometimes called families)

Access Control	Media Protection
Awareness & Training	Personnel Security
Audit & Accountability	Physical Protection
Configuration Management	Risk Assessment
Identification & Authentication	Security Assessment
Incident Response	System & Com Protection
Maintenance	System & Info Integrity

  - What is being assessed – An organization, end user, their systems (as deployed).

Least Privilege

Limit information system access to the types of transactions and functions that authorized users are permitted to execute.

SECURITY REQUIREMENTS	NIST SP 800-53 *Relevant Security Controls*		ISO/IEC 27001 *Relevant Security Controls*	
3.1 ACCESS CONTROL				
Basic Security Requirements				
3.1.1 Limit system access to authorized users, processes acting on behalf of authorized users, or devices (including other systems). 3.1.2 Limit system access to the types of transactions and functions that authorized users are permitted to execute.	AC-2	Account Management	A.9.2.1	User registration and de-registration
			A.9.2.2	User access provisioning
			A.9.2.3	Management of privileged access rights
			A.9.2.5	Review of user access rights
			A.9.2.6	Removal or adjustment of access rights
	AC-3	Access Enforcement	A.6.2.2	Teleworking
			A.9.1.2	Access to networks and network services
			A.9.4.1	Information access restriction
			A.9.4.4	Use of privileged utility programs
			A.9.4.5	Access control to program source code
			A.13.1.1	Network controls
			A.14.1.2	Securing application services on public networks
			A.14.1.3	Protecting application services transactions
			A.18.1.3	Protection of records
	AC-17	Remote Access	A.6.2.1	Mobile device policy
			A.6.2.2	Teleworking
			A.13.1.1	Network controls
			A.13.2.1	Information transfer policies and procedures
			A.14.1.2	Securing application services on public networks
Derived Security Requirements				

More on Authentication

Derived Security Requirements					
3.5.3 Use multifactor authentication for local and network access to privileged accounts and for network access to non-privileged accounts.	IA-2(1)	Identification and Authentication (Organizational Users) *Network Access to Privileged Accounts*	*No direct mapping.*		
	IA-2(2)	Identification and Authentication (Organizational Users) *Network Access to Non-Privileged Accounts*	*No direct mapping.*		
	IA-2(3)	Identification and Authentication (Organizational Users) *Local Access to Privileged Accounts*	*No direct mapping.*		
3.5.4 Employ replay-resistant authentication mechanisms for network access to privileged and non-privileged accounts.	IA-2(8)	Identification and Authentication (Organizational Users) *Network Access to Privileged Accounts-Replay Resistant*	*No direct mapping.*		
	IA-2(9)	Identification and Authentication (Organizational Users) *Network Access to Non-Privileged Accounts-Replay Resistant*	*No direct mapping.*		
3.5.5 Prevent reuse of identifiers for a defined period.	IA-4	Identifier Management	A.9.2.1	User regi de-regist	

SECURITY REQUIREMENTS	NIST SP 800-53 *Relevant Security Controls*		ISO/IEC 27001 *Relevant Security Controls*	
3.5.6 Disable identifiers after a defined period of inactivity.	IA-4	Identifier Management	A.9.2.1	User registration and de-registration
3.5.7 Enforce a minimum password complexity and change of characters when new passwords are created.	IA-5(1)	Authenticator Management *Password-Based Authentication*	*No direct mapping.*	
3.5.8 Prohibit password reuse for a specified number of generations.				
3.5.9 Allow temporary password use for system logons with an immediate change to a permanent password.				
3.5.10 Store and transmit only cryptographically-protected passwords.				
3.5.11 Obscure feedback of authentication information.	IA-6	Authenticator Feedback	A.9.4.2	Secure logon procedures

New Models for Assessment

- There are new assessment methodologies in the works that combine the checklist of controls (e.g. NIST 800-171) with maturity assessment approaches (e.g. CMMI Capability Maturity Model Integration).
  - US DoD Cybersecurity Maturity Model Certification (CMMC)
    - As envisioned levels correspond to the subset of controls implemented AND the highest level of maturity achived.
    - 1 is Basic hygiene, perhaps 17 controls, and ad hoc security processes
    - Level 2 adds controls and required security processed to be documented
    - Level 3 adds more controls and required processes guided by policy
    - Level 5 reqires continuous improvement of process and sharing of data.

Reading Next Lecture

The following readings are linked from the class web page: (ccss.usc.edu/523)

- Attack Trees
- *A Requires/Provides Model for Computer Attacks*
- *Uncover Security Design Flaws Using the STRIDE Approach*
- *Foundations of Attack–Defense Trees*
- *Threat Risk Analysis for Cloud Security based on Attack-Defense Trees*

INF523: Computer System Assurance

Threat Modeling

Prof. Clifford Neuman

Lecture 3
11 Sep 2020

Topics Covered in this Lecture

- Threat modeling techniques and tools
  - Attack Trees and Attack-Defense Trees
  - Requires/Provides modeling
  - Microsoft STRIDE approach and tool

Purpose of Threat Modeling

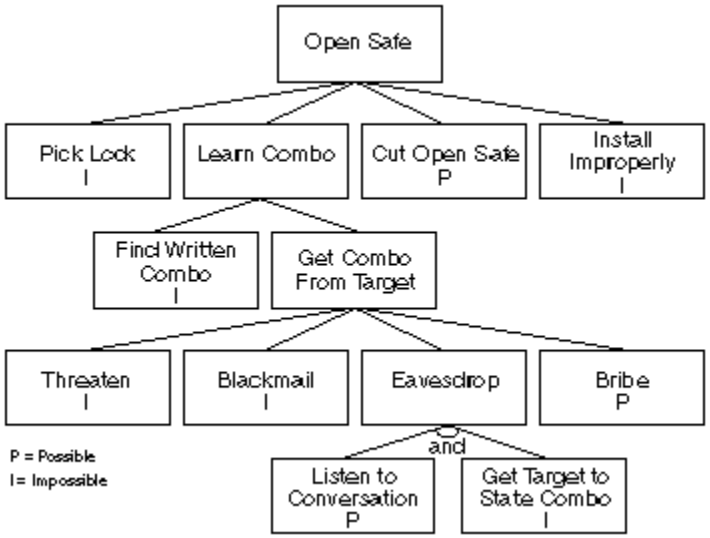
- Identify threats against a system
  - Identify deficiencies in security requirements and design
- Identify threat countermeasures
  - Include, but not limited to, technical mechanisms
  - May include administrative and physical controls
  - Must also consider threats to the countermeasures!
- Increase assurance
- Process should be repeatable, methodical

Attack Trees

- Intended to be a “formal” way of modeling attacks
- “Tree-like representation of an attacker’s goal recursively refined into conjunctive or disjunctive sub-goals”
- Attacker’s “goal” is root of tree
- Different ways of achieving goal are leaves
  - Called “refinements” of the parent goal
- Initially proposed by Schneier in 1999
- Formalized by Mauw and Oostdijk in 2005
  (Foundations of Attack Trees [ICISC’05],
  http://www.win.tue.nl/~sjouke/publications/papers/attacktrees.pdf)

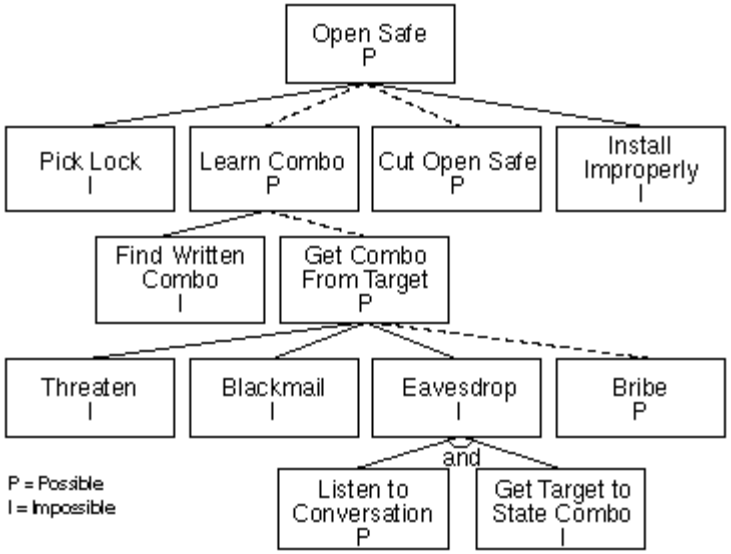
Attack Trees

- Schneier's safe example:
- Mark leaves as "possible" or "impossible".
- "Or" nodes and "and" nodes
- When is goal possible?



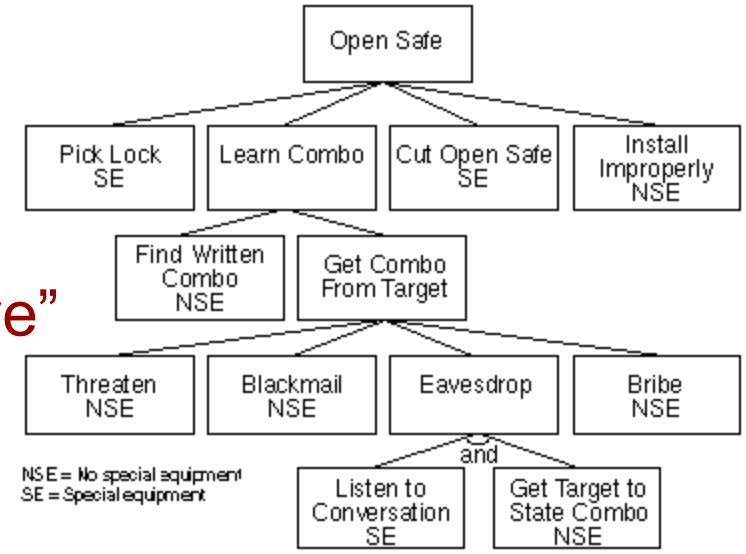
Attack Trees

- Node is “possible” if any of the “or” children beneath it are possible, or if all of the “and” children are possible
- Schneier’s example:



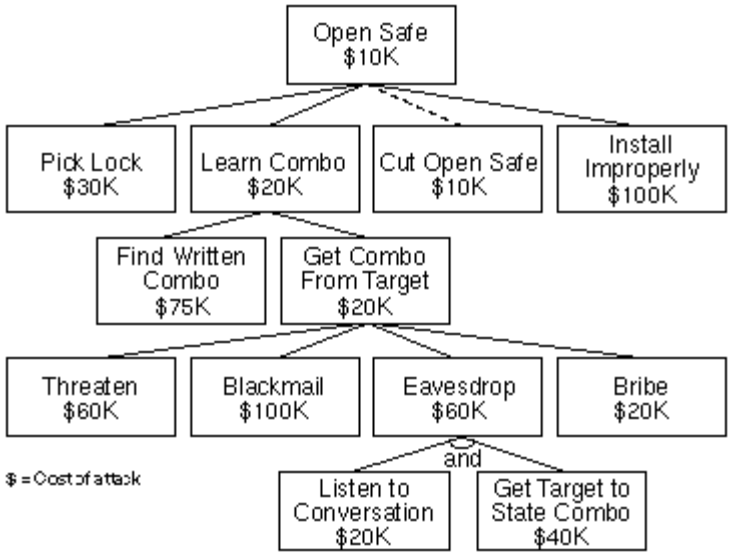
Attack Trees

- Any binary value can be used, not just “possible” and “impossible”
  - Indicates likelihood or risk
  - E.g., “special equipment” vs. “no special equipment”, as in example:
  - “easy” versus “difficult”
  - “expensive” or “cheap”
  - legal versus illegal
  - “intrusive” versus “nonintrusive”



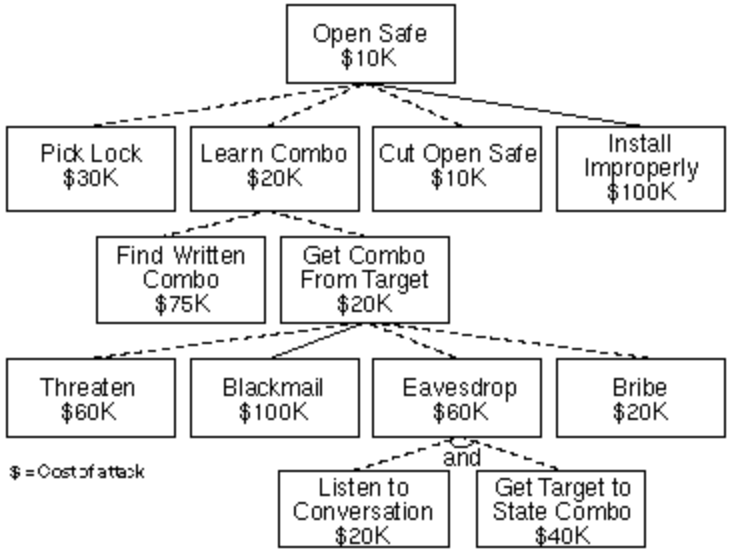
Attack Trees

- Can also use a continuous value function
- E.g., Schneier's example:
- Estimated cost to attacker of each refinement
- Value in each node is sum of "and" leaves or lowest value of "or" leaves
  - Assumes cost isimportant factor for attacker



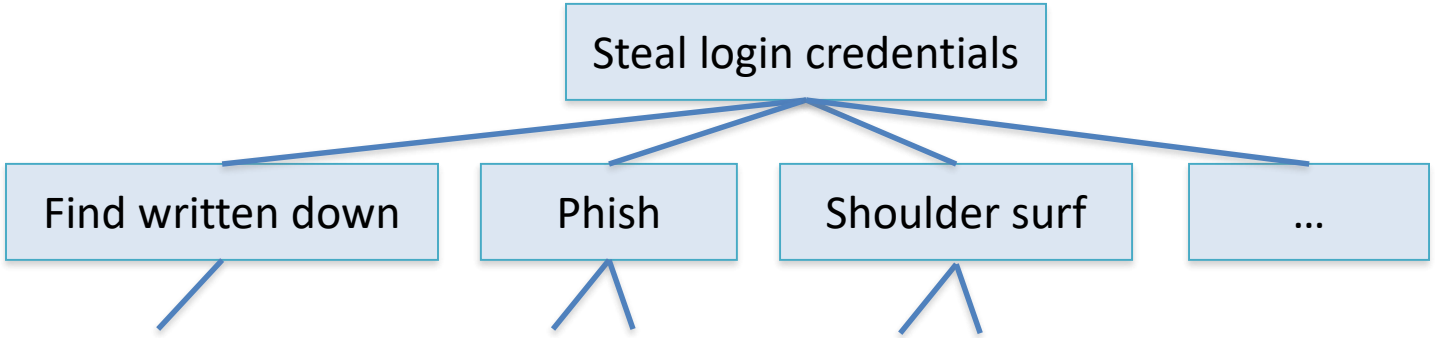
Attack Trees

- As with boolean values, continuous functions used to indicate likelihood or risk of particular attack
- Can combine multiple functions
  - E.g., “cheapest attack with th success”



Attack Tree Exercise

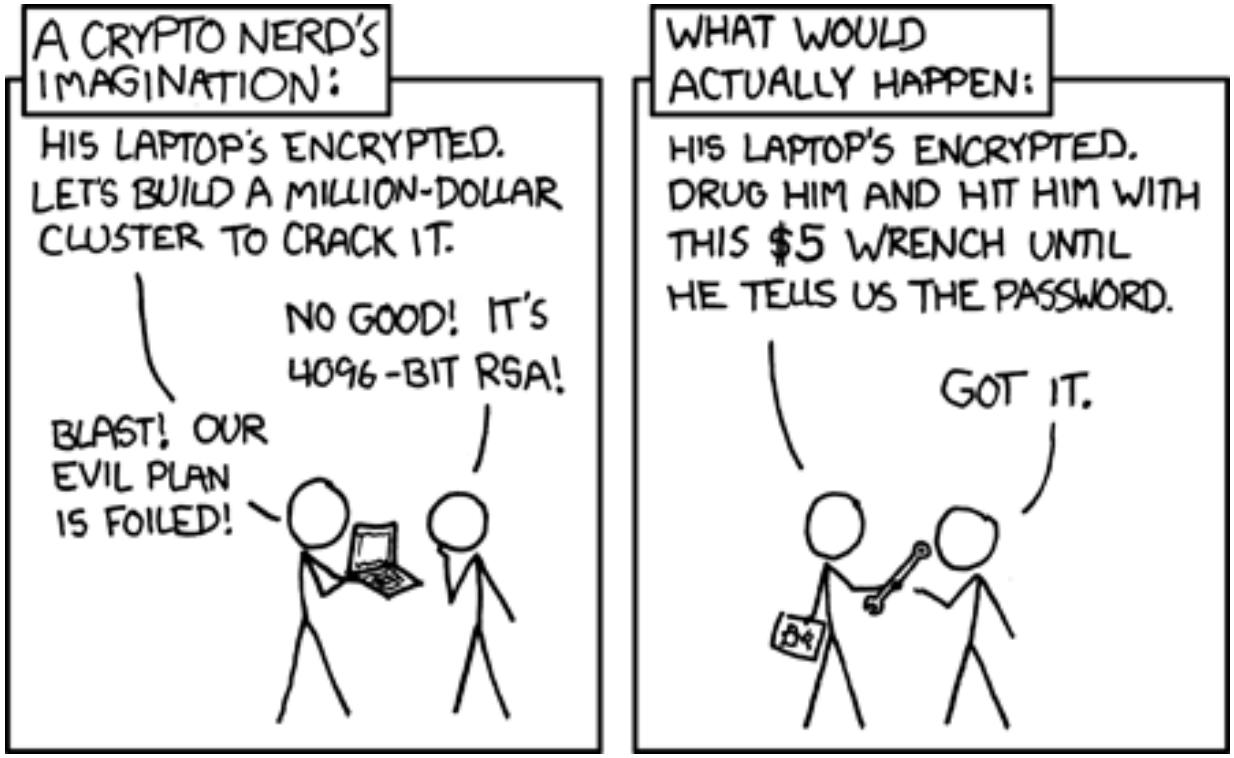
- Continue to fill out this attack tree



Attack Trees

- Knowledge and creativity needed by analysts
  - Think like an attacker
  - All sorts of vulnerabilities in different sub-systems
  - Analysts must understand all parts of the system well
- How do you determine a good value for a leaf node?
  - Analyst must study presumed attackers as well
  - E.g., if organized international crime, have lots of money, expertise and little fear of jail, so what is threshold?
- Often highly subjective

Attack Trees



https://xkcd.com/538/

Countermeasures

- Once tree “complete”, use it to identify countermeasures
- Bring value of node below threshold to “deactivate”
  - E.g., a countermeasure that makes a leaf “impossible”
  - Or that makes too expensive
- Do that for all “or” leaves or any “and” leaf to deactivate parent
- Recurse up the tree to root

Attack Trees, Pros and Cons

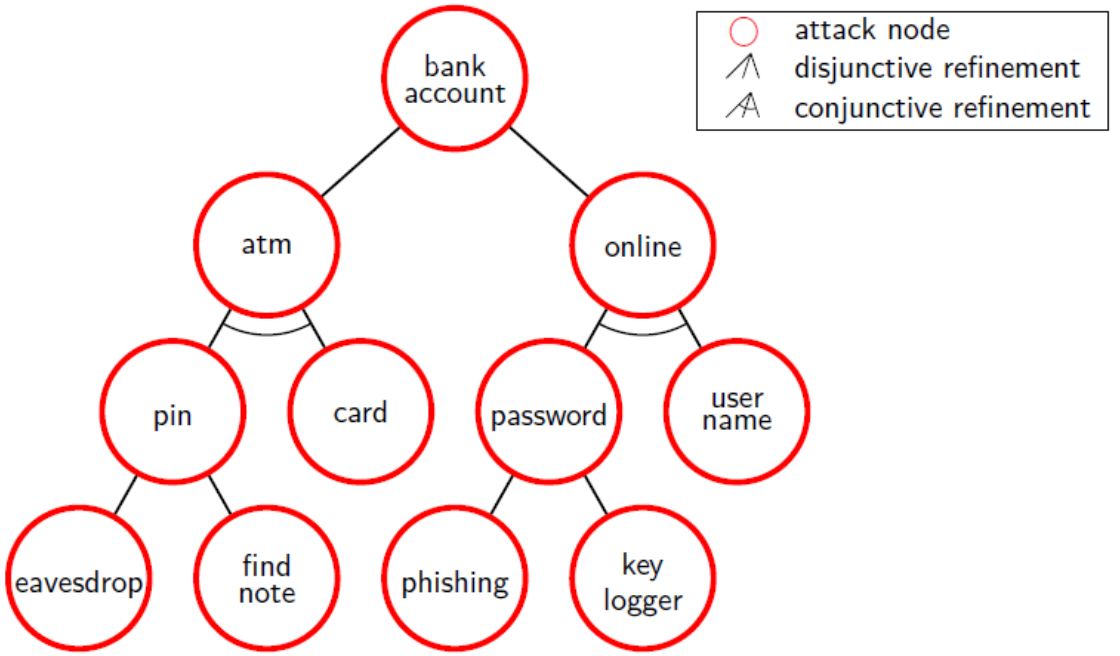
- Pros
  - Conceptually simple
  - Scalable
  - Reusable
- Cons
  - Only considers attacker's point of view
  - No countermeasures in the graph
    - How do you show attacks on the countermeasures?
  - No attacker/defender interactions
  - Simple signatures or single-point exploits
  - Weak or no explicit link between steps
    - How are they related? Ordering?

Attack-Defense Trees

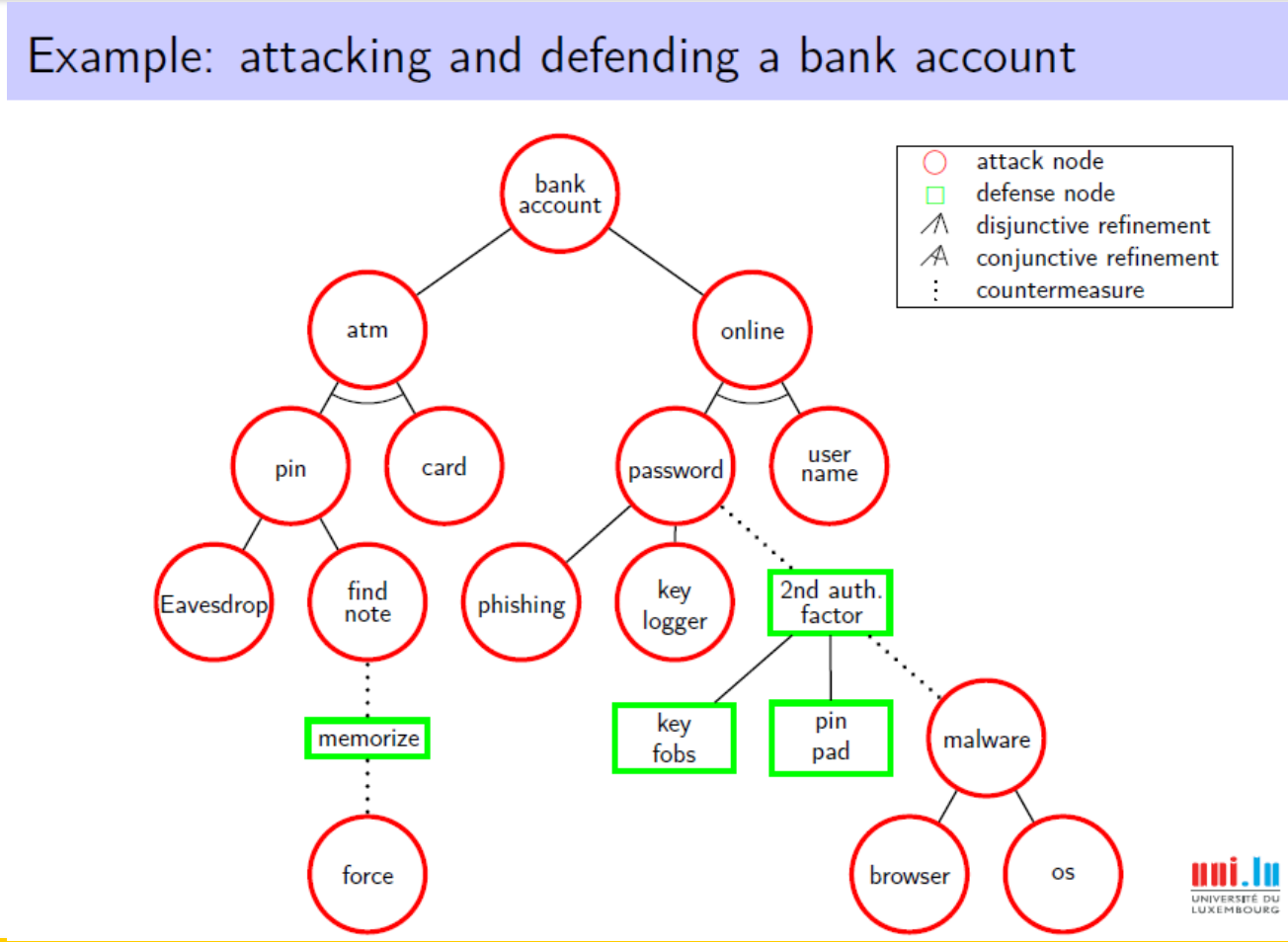
- Introduced by Kordy et al.
  (*Foundations of Attack–Defense Trees [FAST'10],*
  http://satoss.uni.lu/members/barbara/papers/adt.pdf)
- Includes countermeasures, so can show attacks on countermeasures

Attack-Only Tree Example

Example: attacking a bank account



Attack-Defense Tree Example

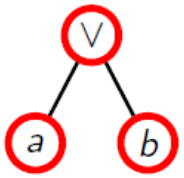


Attack-Defense Trees

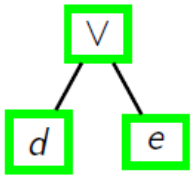
ADTrees as Boolean functions

$f(a) = a$

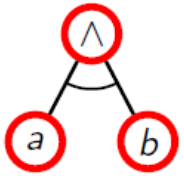
$f(d) = d$



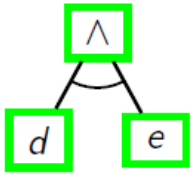
$f(a, b) = a \vee b$



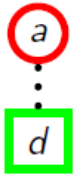
$f(d, e) = d \vee e$



$f(a, b) = a \wedge b$



$f(d, e) = d \wedge e$



$f(a, d) = a \wedge \neg d$

$f(d, a) = d \wedge \neg a$

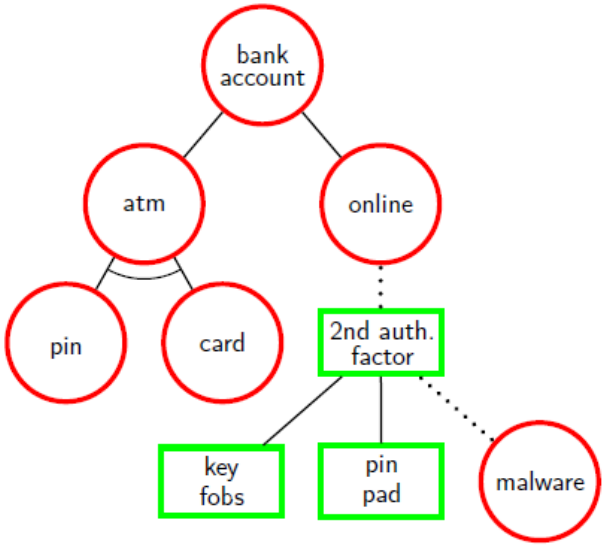
Back

Attack-Defense Tree Example

Example: propositional interpretation of an ADTree

DeMorgan's law => f = (pin /\ card) \/ (online /\ (~(key fobs \/ pin pad) \/ malware))

$$f = (\text{pin} \wedge \text{card}) \vee \Big(\text{online} \wedge \neg((\text{key fobs} \vee \text{pin pad}) \wedge \neg\text{malware})\Big)$$

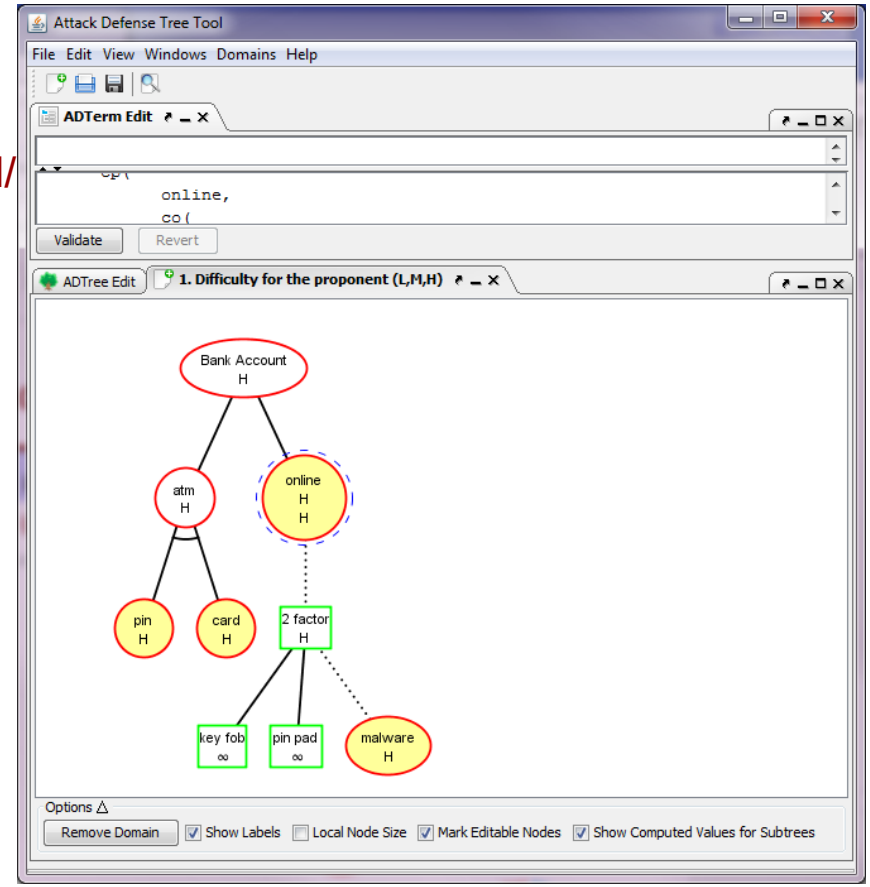


Details

uni.lu UNIVERSITÉ DU LUXEMBOURG

ADTool

- Tool for supporting the ADTree methodology
  - http://satoss.uni.lu/members/piotr/adtool/
- Let's try it out



A-D Trees

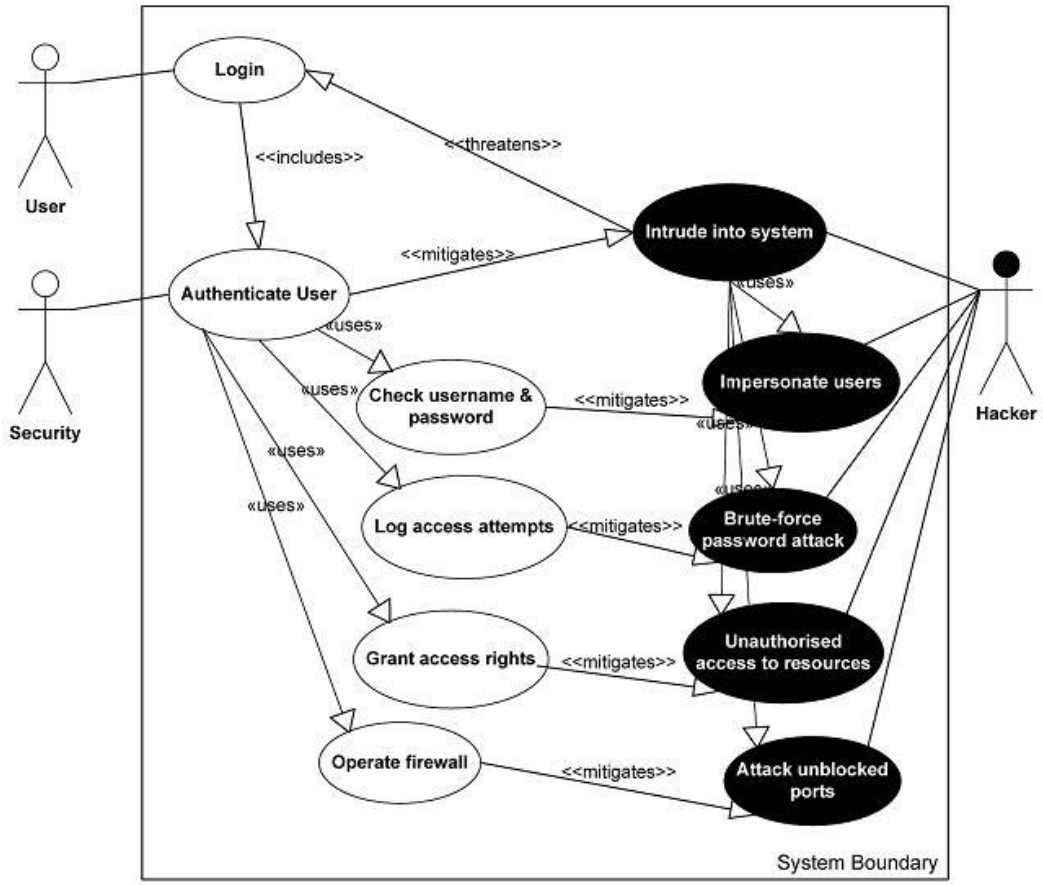
- Pros
  - Conceptually simple, but not as simple as plain trees
  - Scalable (assuming you don't go hog-wild with the countermeasures)
  - Reusable
  - Consider defender's POV as well as attacker's
  - Incorporates countermeasures and attacks on countermeasures
- Cons
  - Simple signatures or single-point exploits
  - Weak or no explicit link between steps
    - How are they related? Ordering?

Unified Modelling Language (UML)

- Language for specifying, visualizing, constructing, and documenting models for systems
- A *set of notations*, not a model itself
- Different diagram types:
  - Use case, Class, Activity, Collaboration, Sequence, State, …
  - For more info, http://www.uml-diagrams.org/

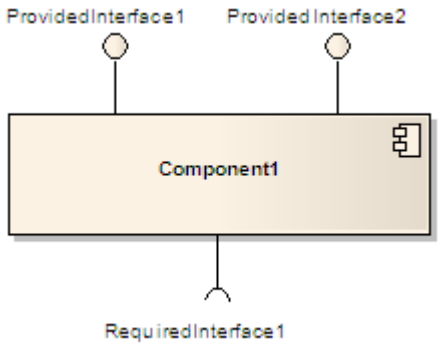
Example of UML Use Case Diagram

- Actors
- Associations
- Relationships



UML Component Diagrams

- Activities
  - Tasks that must take place in order to "fulfill operational contract"
  - Invocation of operations
  - Steps in processes or entire process
  - Can decompose down to atomic actior
- Components of the system
  - Modules, but with "required" and "provided" interfaces
- How the components interact
  - Component diagram shows wiring



Requires/Provides Model

- Templeton and Levitt, 2000

Single Exploits vs. Sequence

- Single exploit
  - Short term goal
  - May or may not violate some part of the security policy
  - E.g., a port scan
- Sequence of single exploits (scenario)
  - Has an end goal in mind
  - Explicitly violates security policy
  - E.g., port scan followed by buffer overflow followed by installation of back door …
  - Very dangerous

Generalized Sequences of Attacks

- Port scan followed by buffer overflow followed by installation of back door is very specific
- More general, *recon* followed by *exploit* followed by *penetration*
  - *Exploit* depends on knowledge gained by *recon*
  - *Penetration* depends on capability gained by *exploit*
- Want to abstractly model attacks based on
  - the requirements of the abstract components,
  - the capabilities provided by the abstract components, and
  - the method of composing the components into complete attacks

Requires/Provides Model

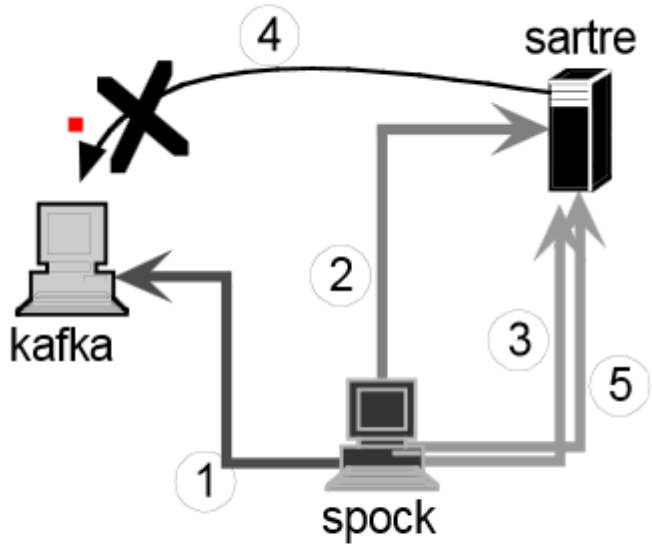
- To successfully launch an attack, certain properties must hold
  - These are the *requires* properties
- After a successful attack, a new set of properties hold
  - These are the *provides* properties
- The attack "goal" is a property that holds after a sequence of attack events

Example Attack Sequence

- Kafka has rsh access on sartre
- Spock wants to run code on sartre

1. Spock DoSes kafka with flood
2. Spock probes sartre for TCB seq num
3. Spock sends spoofed SYN packet (as kafka)
4. Sartre sends to kafka, which is blinded
5. Spock sends rsh packet to sartre

Figure #1: RSH Connection Spoofing



1: synflood initiated on kafka
2: spock probes sartre for starting sequence number info
3: spock sends spoofed packet to sarte
4: sartre responds to kafka, but kafka is not responding
5: spock sends spoofed packet sartre expected to get from kafka. data includes body of attack.

Connection Spoofing R/P

- Requires:
  - "Trustor" running active service (Sartre)
  - Trusted partner (pretend to be trusted partner) (kafka)
  - Ability to prevent trusted partner from receiving
  - Ability to probe trustor for TCB sequence number
  - Ability to send a forged packet
- Provides:
  - Ability to send data to trusted channel
  - Ability to have data remotely executed
- These are general properties
- Instantiate for rsh or other protocols

Similarity to Attack Trees

- Goal: Get Sartre to execute commands from untrusted host Spock
- Sub-goal: Get Sartre to believe trusted host Kafka is sending the commands
  - Must prevent ACK from Sartre from reaching Kafka
  - Must determine what sequence number Sartre would use, so Spock can use that in “response” to blocked ACK

- But different from attack trees in specifying order

Creating Variant Attacks

- Different events can cause the same effects
- Different orderings of events can cause the same effects
- Want to reason in terms of the effects of an event, not on the details of an event itself
  - E.g., instead of SYN-flood, the attacker on Spock could have use a packet storm, ping-of-death, or even physically disabled the network cable to Kafka
  - Each of these would have had the same effect of blocking Kafka from receiving ACKs from Sartre

Concepts and Capabilities

- *Capabilities* are the (generalized) information or situation required for an attack to proceed
  - E.g., User login requires access, user name, password
  - System requires access to password validation database
  - Atomic elements of the model
  - Generalized capability is template for instantiations
- *Concepts* map required capabilities to provided capabilities and instantiate capabilities
- Attacks are defined as the composition of abstract *concepts*

Inherent Implication

- Existence of a capability implies existence of another
  - E.g., A prevented from sending a packet to B ➔ B is prevented from receiving a packet from A
  - B is prevented from receiving a packet from A ➔ B is prevented from sending reply packet back to A
- Don't depend on implication
- Must explicitly state concepts that define each implication

JIGSAW

- Language developed to express capabilities and concepts
- Permits mechanization
  - Can automatically discover ways that capabilities can be combined into attacks
- Capability templates
  - Named collection of typed attribute-value pairs
- Concepts
  - Set of required and provided capabilities
  - “With” section gives relations that must hold between the required capabilities

Example Capability

capability *Trusted_Partner* **is**
service: ***service_type****;*
trustor: ***ip_addr_type****;*
trusted: ***ip_addr_type***;
end.

Example Concept (abbreviated)

Concept RSH_Connection_Spoofing

requires

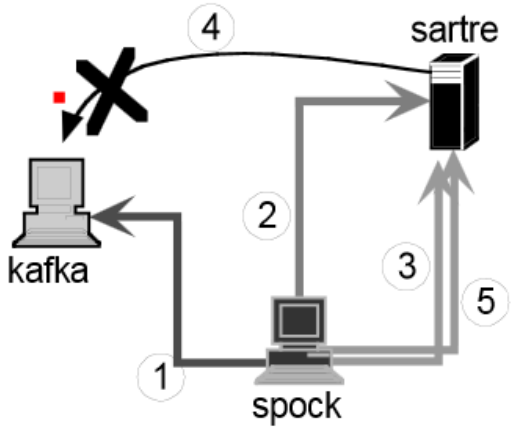
Trusted_Partner: TP;
ForgedPacketSend: FPS
PreventPacketSend: PPS;
…

with

TP.service is RSH,
PPS.host is TP.trusted,
FPS.dst.host is TP.trustor,
…

end;

Figure #1: RSH Connection Spoofing



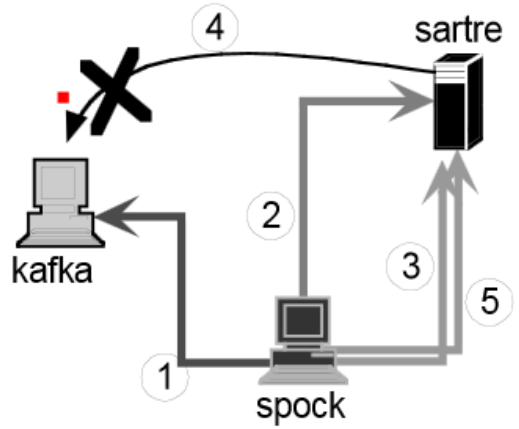
1: synflood initiated on kafka
2: spock probes sartre for starting sequence number info
3: spock sends spoofed packet to sarte
4: sartre responds to kafka, but kafka is not responding
5: spock sends spoofed packet sartre expected to get from kafka. data includes body of attack.

Example Concept (abbreviated)(cont.)

Concept RSH_Connection_Spoofing, continued

```
provides
    push_channel:          PSC;
    remote_execution:  REX;
with
    PSC.from          <- FPS.true_
    PSC.to          <- FPS.dst;
    PSC.using          <- RSH;
    REX.from      <- FPS.true_src
    …
end;
```

Figure #1: RSH Connection Spoofing



1: synflood initiated on kafka
2: spock probes sartre for starting sequence number info
3: spock sends spoofed packet to sarte
4: sartre responds to kafka, but kafka is not responding
5: spock sends spoofed packet sartre expected to get from kafka. data includes body of attack.

Power of Model

- Ordering and relationship of attack steps implicit in that *provides* must precede *requires*
  - Compare to attack trees
  - Capabilities essentially form edges of R/P attack graph
- Multiple events can provide equivalent capabilities
- Attack scenarios can have many variants
  - instantiate different events/protocols that provide same capabilities
- Exploits can be combined in new ways to create previously unexpected attacks
  - Just have to satisfy capabilities

Weakness of RP

- A technique for modelling multi-step abstract attacks
- No connection to policy (same as attack trees)

Microsoft STRIDE Model

- Developed by Microsoft and refined over the last 10 years
- Applied to all software development activities

Microsoft's Software Security Properties

Property	Description
Confidentiality	Data is only available to the people intended to access it.
Integrity	Data and system resources are only changed in appropriate ways by appropriate people.
Availability	Systems are ready when needed and perform acceptably.
Authentication	The identity of users is established (or you're willing to accept anonymous users).
Authorization	Users are explicitly allowed or denied access to resources.
Nonrepudiation	Users can't perform an action and later deny performing it.

STRIDE

- Acronym for categories of threats:

Threat	**Security Property at Risk**
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

Meaning of Each Threat Class

- **Spoofing** : Impersonating something or someone else
- **Tampering** : Modifying data or code
- **Repudiation** : Claiming to have not performed an action
- **Information Disclosure** : Exposing information to someone not authorized to see it
- **Denial of Service** : Deny or degrade service to users
- **Elevation of Privilege** : Gain capabilities without proper authorization

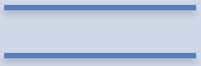
STRIDE Steps

- Decompose system into components
  - May need to recurse down to necessary level of detail
- Analyze each component for susceptibility to each relevant type of threat
- Develop countermeasures until no component has susceptibility
- Is system secure?
  - Maybe, but probably not
  - Due to emergent properties of composition
- Does this give higher assurance?
  - Yes, because flaw in one component affects entire system

Data Flow Diagram (DFD)

- Used to graphically represent a system and its components
- Standard set of elements:
  - Data flows
  - Data stores
  - Processes
  - Interactors
- One more for threat modeling:
  - Trust boundaries

DFD Symbols

Element	Shape	Description
Process		Any running computations or programs
Interactor		A user, service, or machine that interacts with the application and is external to it – either as a data producer or consumer
Data Store		Any data "at rest" on some form of storage (e.g., files, DBs, registry keys, etc.)
Data Flow		Any transfer of data from one element to another (via network, pipe, RPC, etc.)
Trust Boundary		Border between "trusted" and "untrusted" elements

Relevant Threats for Elements

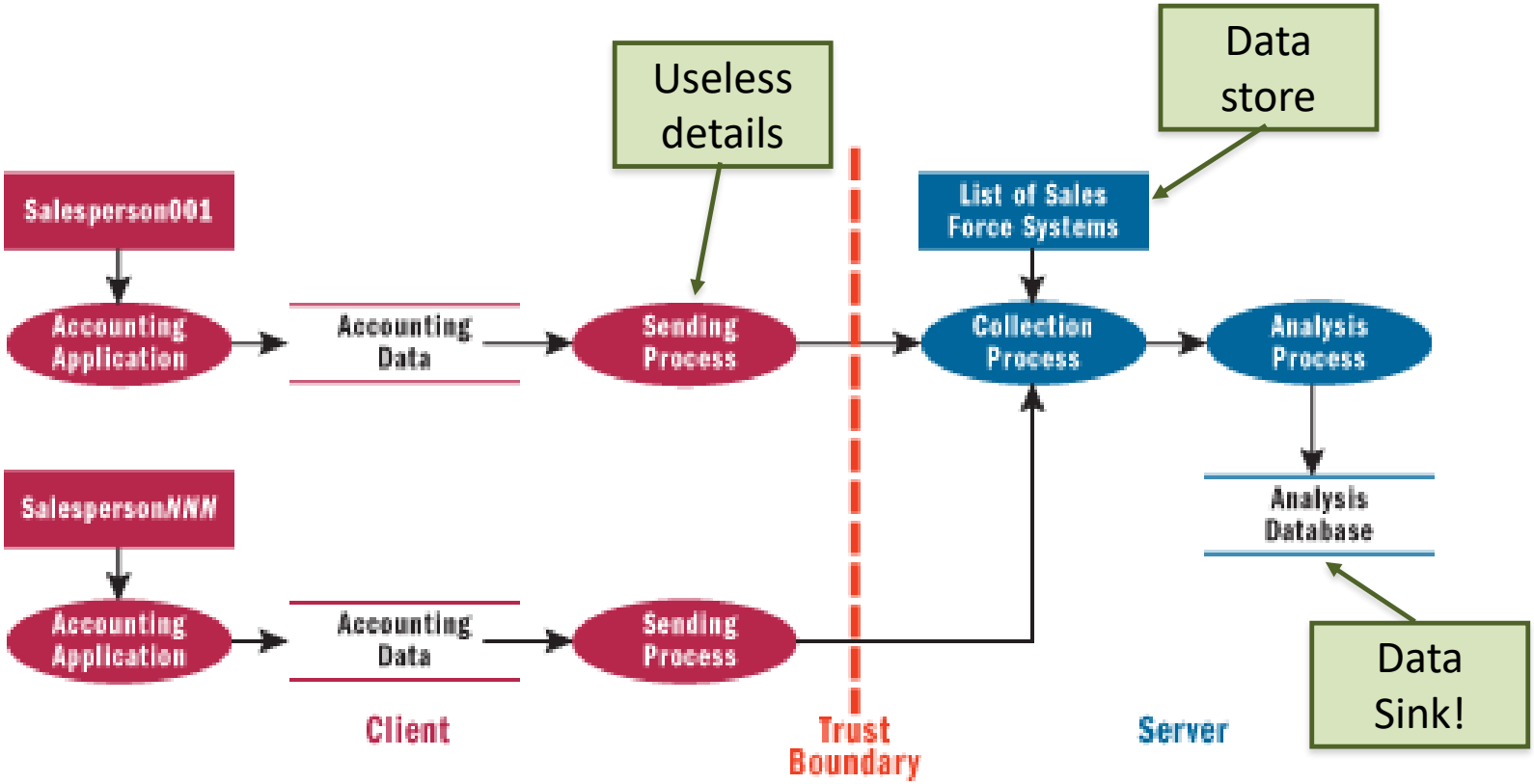
	Interactors	Process	Data Store	Data Flow
Spoofing	x	x		
Tampering		x	x	x
Repudiation	x	x	*	
Information disclosure		x	x	x
Denial of Service		x	x	x
Elevation of Privilege		x		

* Logs held in data stores are usually the mitigation against a repudiation threat. Data stores often come under attack to allow for a repudiation attack to work.

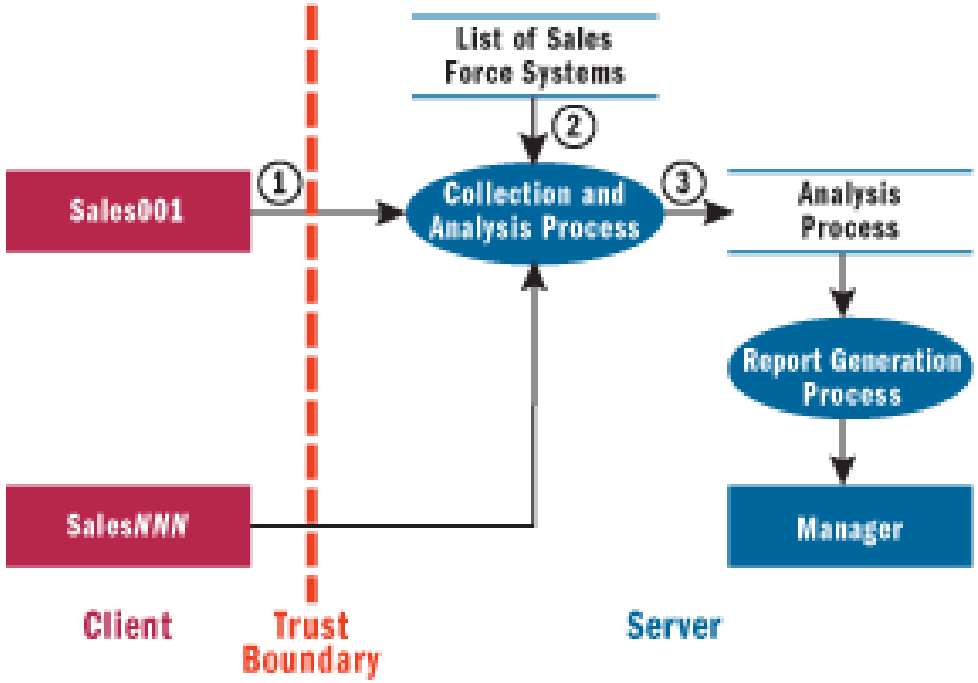
STRIDE Process

- Create DFD of system
  - Represent all key components
  - Represent all data flows
  - Identify trust boundaries
- Repeat, adding more details to the diagram if required
- Recurse on each component as required

Example: First Cut



Example: Second Try



1. Sales001 ->Collection and Analysis Process
2. List of Sales Force Systems->Collection and Analysis Process
3. Collection and Analysis Process-> Analysis Process

3 data flows

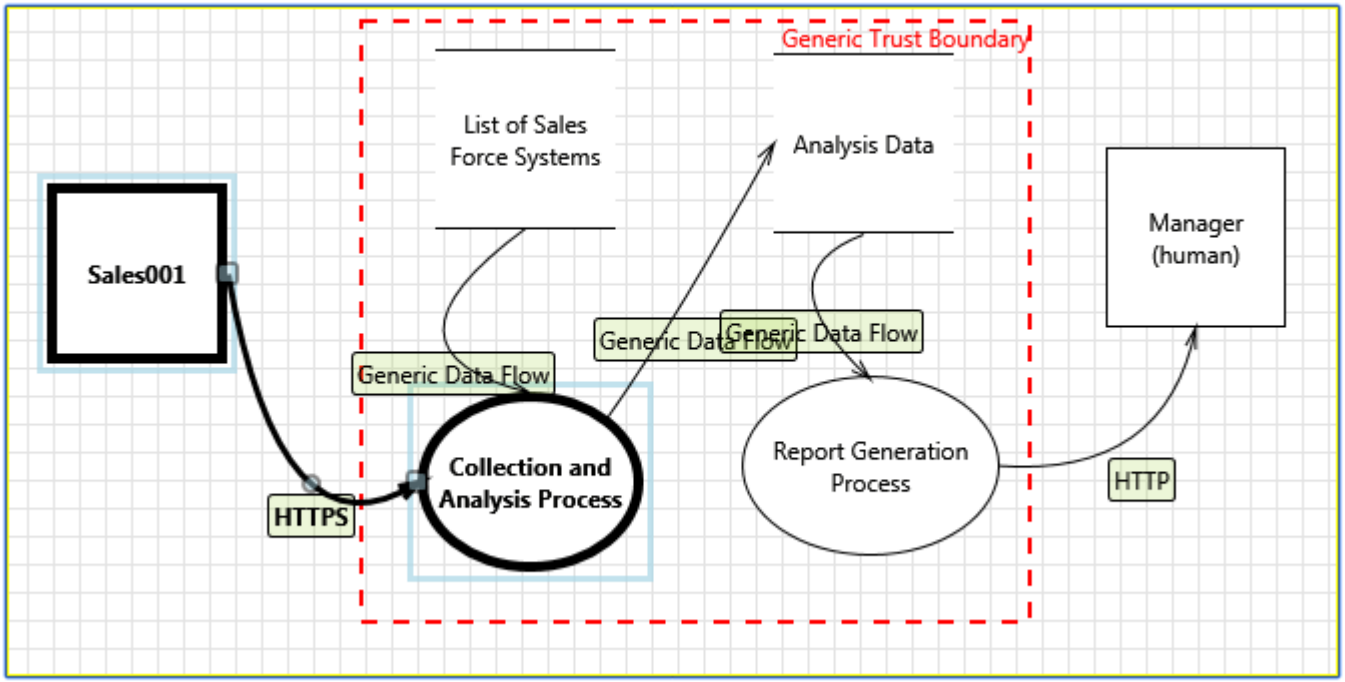
Analysis: Data Flow 1

- Sales to Collection
- Someone could tamper with the data in transit
- Someone could sniff the data
- Someone could DoS the collection service

	Data Flow
Spoofing	
Tampering	x
Repudiation	
Information disclosure	x
Denial of Service	x
Elevation of Privilege	

MS Threat Modeling Tool 2014

- Software for applying STRIDE model
  - Build DFD directly in program
  - Automatically finds STRIDE threats



Mitigate Threats

- Tool has places to specify status of mitigation:
  - Not Started
  - Needs Investigation
  - Not Applicable
  - Mitigated
- If you say Mitigated or Not Applicable, must enter Justification
- Also can select priority (Low, Medium, High)
  - Used for the “bug bar” (ranking of threats by priority)
  - E.g., see http://msdn.microsoft.com/en-us/library/windows/desktop/cc307404.aspx

Controls to Mitigate Threats

- Remove vulnerable feature
- "Fix" with technology, e.g.:
  - Spoofing
    - Strong authentication
  - Tampering
    - Strong authorization (restrict modify access)
  - Repudiation
    - Digital signatures, timestamps
  - Information Disclosure
    - Encryption
  - Denial of Service
    - Packet filtering
  - Elevation of Privilege
    - Restrict admin privilege

Mitigation Choices in Reality

- Redesign
  - Change the design to eliminate threats
  - E.g., reduce elements that touch a trust boundary
- Use standard mitigations
  - Firewalls, validated authentication systems, …
- Use custom mitigations
  - If you are a gambling sort of person
- Accept risk
  - If you think risk is low, or too expensive to mitigate

Validation

- Make sure diagram is up-to-date and accurate
- Make sure you've captured all trust boundaries
- Enumerate all threats
  - The tool is an aid, but not necessarily complete
- Analyze all threats
- Mitigate all threats

Diagram Layers

- Context Diagram
  - Very high-level; entire component / product / system
- Level 1 Diagram
  - High level; single feature / scenario
- Level 2 Diagram
  - Low level; detailed sub-components of features
- Level 3 (, 4,…) Diagram
  - More detailed yet, if necessary

Combine STRIDE With Other Techniques

- Use UML instead of DFD to determine threat targets
- Determine threats to each component using STRIDE
- Use threat trees to help determine vulnerabilities
  - Each STRIDE threat is the root of a tree
- Use a risk assessment method to rank threats

STRIDE Pros and Cons

- STRIDE identifies security properties and threats against them
  - *Confidentiality*, *Integrity*, Availability, Authentication, Authorization, Nonrepudiation
  - Those are effectively security policies
  - But where in the model are all those Windows bugs?
    - And IE bugs
- Are threats comprehensive?
- Patch and pray school of system design
- No reference monitor concept for access policies
  - Better to try to design RM, then look for threats
    - To isolation, completeness, verifiability

Topics Covered so far in this Lecture

- Threat modeling techniques and tools
  - Attack Trees and Attack-Defense Trees
  - Requires/Provides modeling
  - Microsoft STRIDE approach and tool

Security Requirements

- Many different definitions/approaches
  - E.g., Square, Clasp, STRIDE, …
    - See "Security Requirements for the Rest of Us: A Survey", *IEEE Software*, January/February 2008
- Differences:
  - Security mechanisms or policy?
  - Level of detail?
  - Level of expert knowledge?

Factors in Determining Security Requirements

- Organizational requirements
  - Hopefully based on well-defined policy
- Sometimes to counter specific threats
  - E.g., MS STRIDE tool:
    - Spoofing - Strong authentication
    - Tampering - Strong authorization (restrict modify access)
    - Repudiation - Digital signatures, timestamps
    - Information Disclosure - Encryption
    - Denial of Service - Packet filtering
    - Elevation of Privilege - Restrict admin privilege
- Regulations or laws

Example of Reqs due to Law or Regulation

- HIPAA 45 CFR 164.312 - Technical safeguards (e)
  - **(1) *Standard: Transmission security.*** Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.
  - **(2) *Implementation specifications:***
    - **(i)** *Integrity controls* (Addressable). Implement security measures to ensure that electronically transmitted electronic protected health information is not improperly modified without detection until disposed of.
    - **(ii)** *Encryption* (Addressable). Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate.

"Addressable" means alternative may be used if requirement is unreasonable or inappropriate.

Other Factors

- Costs
- Priority based on
  - Threat actors
    - Goals
    - Expertise
    - Resources
  - Likelihood of attack
  - Degree of difficulty for attacker
  - Value (estimated loss)
  - "L", "M", "H" is probably best resolution possible

Accurate estimates of these are likely impossible!

Security Requirements Chain

- "Security objectives" (org reqs) + threats
  - Lead to policy
  - Don't forget subversion!
- Policy defines
  - what assets need protection
  - what "security" means for the assets
  - "Security" in terms of secrecy and integrity
    - Also identification/authentication, audit, authorization
    - May also be availability, non-repudiation, etc.
- Policy leads to mechanisms to enforce the policy
- Which are the security requirements?

Policies or Mechanisms; Which are the Security Reqs?

- It depends on the organization
- If an organization doesn't have a security policy:
  - Have no choice but to include policy in reqs
- If an organization has a security policy:
  - Reqs as in HIPAA are a good level
  - But where do you specify mechanisms? Who builds them?
  - Developers often have no security experience or interest
- Role of security analyst sometimes comprehensive
  - Specify requirements
  - *and* enforcement mechanisms
  - *and* verification of the enforcement mechanisms

Security Requirements - Mechanisms

- Policy reqs lead to mechanisms to enforce policy
- Reference Monitor concept useful here
  - Rather than scattershot reqs on system components
  - Mediate access by subjects to objects
  - Attempt to implement isolation, completeness, verifiability
- Look for threats against isolation and completeness
  - E.g., transmitted data could be sniffed
- Mechanisms counter the threats to the RM
  - E.g., in network, encrypt data in transit
  - TNI can be helpful

Implementing Security Requirements

- Must trust mechanisms to enforce policy
- Structured design helps to provide assurance
- The subject of today's lecture

Homework

- Due next week at start of class
  - Submit screen shots and other documents on D2L
- Remember, you can help each other understand the assignment, the concepts, and the tools
  - But the work you turn in must be your own

- Analyze threats to a simple on-line payment system

Homework Problem

- A (simple) on-line payment system runs on a web server
- Users connect using a web browser via HTTPS
- Users authenticate using passwords
- The server runs the payment application
- The application consults a back-end authorization database
- The application connects to a back-end DB server to record payments
- The DB server stores credit card information
- An attacker wants to steal credit card information

Homework

1. Create a plain attack tree
   - Use “hard” and “easy” as node values
   - What is the easiest route?
2. Create a corresponding A-D tree
   - Use ADTool (requires Java 6 or later)
     - http://satoss.uni.lu/members/piotr/adtool/
   - Include defensive measures and attacks on defensive measures
   - Give the propositional interpretation of the tree
3. Write-up R/P capabilities and a concept for an attack on this system via the web connection
4. Create a STRIDE threat model
   - Show all processes, interactors, stores, flows, and boundaries
   - Use Threat Modeling Tool if you have a Windows machine
   - Identify threats and some countermeasures

Reading for Next Time (Software Design)

- D.L. Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules,* 1972
- Daniel Hoffman, *On Criteria for Module Interfaces*, 1990
- Paul Karger, et. al., *A VMM security kernel for the VAX architecture*, 1990 – Section 3.7
- Final Evaluation Report, Gemini Trusted Network Processor, 1995 – Section 4.2