Evaluating System Security
Computer Security
Peter Reiher
May 31, 2016

Evaluating Program Security

- What if your task isn't writing secure code?
- It's determining if someone else's code is secure
  - Or, perhaps, their overall system
- How do you go about evaluating code or a working system for security?

Secure System Standards

- Several methods proposed over the years to evaluate system security
- Meant for head-to-head comparisons of systems
  - Often operating systems, sometimes other types of systems
  - Usually for HW/SW, not working systems

define a standard that say: if you built your software using this standard, you can have this property.

Some Security Standards

- U.S. Orange Book
- Common Criteria for Information Technology Security Evaluation
- There were others we won't discuss in detail

The U.S. Orange Book

- The earliest evaluation standard for trusted operating systems
- Defined by the Department of Defense in the late 1970s
- Now largely a historical artifact

Purpose of the Orange Book

- To set standards by which OS security could be evaluated
- Fairly strong definitions of what features and capabilities an OS had to have to achieve certain levels
- Allowing "head-to-head" evaluation of security of systems
  - And specification of requirements

Orange Book Security Divisions

- A, B, C, and D
  – In decreasing order of degree of security
- Important subdivisions within some of the divisions
- Required formal certification from the government (NCSC)
  – Except for the D level

Why Did the Orange Book Fail?

- Expensive to use
- Didn't meet all parties' needs
  - Really meant for US military
  - Inflexible
- Certified products were slow to get to market
- Not clear certification meant much
  - Windows NT was C2, but that didn't mean NT was secure in usable conditions
- Review procedures tied to US government

assurance that your product is secure; NT certified up to C2; didn't mean NT was secure in usable conditions.

sample question: why did the orange book fail?

The Common Criteria

- Modern international standards for computer systems security
- Covers more than just operating systems
  - Other software (e.g., databases)
  - Hardware devices (e.g., firewalls)
- Design based on lessons learned from earlier security standards
- Lengthy documents describe the Common Criteria

set up a body that is going to define some standards

Common Criteria Approach

- The CC documents describe
  - The Evaluation Assurance Levels (EAL)
    - 1-7, in increasing order of security
- The Common Evaluation Methodology (CEM) details guidelines for evaluating systems
- PP – Protection Profile
  - Implementation-independent set of security requirements

specific to the product or needs people have; if someone is more interested in integrity, and someone is more devoted to security, then we need to accomodate for them.

Another Bowl of Common Criteria Alphabet Soup

- TOE – Target of Evaluation
- TSP – TOE Security Policy
  – Security policy of system being evaluated
- TSF – TOE Security Functions
  – HW, SW used to enforce TSP
- ST – Security Target
  – Predefined sets of security requirements

policies are: this is what I want to achive; and you need functions
in order to work out perfectly.

What's the Common Criteria About?

- Highly detailed methodology for specifying :
  1. What security goals a system has?
  2. What environment it operates in?
  3. What mechanisms it uses to achieve its security goals?
  4. Why anyone should believe it does so?

How Does It Work?

- Someone who needs a secure system specifies what security he needs
  - Using CC methodology
  - Either some already defined PPs
  - Or he develops his own
- He then looks for products that meet that PP
  - Or asks developers to produce something that does

How Do You Know a Product Meets a PP?

- Dependent on individual countries
- Generally, independent labs verify that product meets a protection profile
- In practice, a few protection profiles are commonly used
- Allowing those whose needs match them to choose from existing products

Status of the Common Criteria

- In wide use
- Several countries have specified procedures for getting certifications
  - Some agreements for honoring other countries' certifications
- Many products have received various certifications

Problems With Common Criteria

- Expensive to use
- Slow to get certification
  - Certified products may be behind the market
- Practical certification levels might not mean that much
  - Windows 2000 was certified EAL4+
  - But kept requiring security patches . . .
- Perhaps more attention to paperwork than actual software security
  - Lower, commonly used EALs only look at process/ documentation, not actual HW/SW

Evaluating Existing Systems

- Standards approaches aren't always suitable
- Not helpful for evaluating the security of running systems
- Not great for custom systems
- What do you do for those problems?

Two Different Kinds of Problems

1. I need to evaluate the design and implementation of the system
2. I need to evaluate what's going on in the system as it runs

Evaluating System Design Security

- Sometimes standards aren't the right choice
- What if you're building your own custom system?
- Or being paid to evaluate someone else's?
  - That's some companies' business
- This kind of review is about design and architecture
  - Evaluating running systems comes later

How Do You Evaluate a System's Security?

- Assuming you have high degree of access to a system
  - Because you built it or are working with those who did
- How and where do you start?
- Note: there are many different approaches
- Much of this material is from "The Art of Software Security Assessment," Dowd, McDonald, and Schuh

Stages of Review

- You can review a program's security at different stages in its life cycle
  - During design
  - Upon completion of the coding
  - When the program is in place and operational
- Different issues arise in each case

Design Reviews

- Done perhaps before there’s any code
- Just a design
- Clearly won’t discover coding bugs
- Clearly could discover fundamental flaws
- Also useful for prioritizing attention during later code review

Purpose of Design Review

- To identify security weaknesses in a planned software system
- Essentially, identifying threats to the system
- Performed by a process called *threat modeling*
- Usually (but not always) performed before system is built

Attack Surfaces

- Attackers have to get into your software somehow
- The more ways they can interact with the software, the more things you must protect
- Some entry points are more dangerous than others
  - E.g., those that lead to escalated privilege
- A combination of these factors defines a system's *attack surface*
- The smaller the attack surface, the better
  - But attack surface doesn't indicate actual flaws, just places where they could occur

not all entry points are the same danger; somone easierser to exploirn

small attack surfaces can be deadliu wellwell

Threat Modeling

- Done in various ways
- One way uses a five step process:
  1. Information collection
  2. Application architecture modeling
  3. Threat identification
  4. Documentation of findings
  5. Prioritizing the subsequent implementation review

1. Information Collection

- Collect all available information on design
- Try to identify:
  - Assets
  - Entry points
  - External entities
  - External trust levels
  - Major components
  - Use scenarios

One Approach[1]

- Draw an end-to-end deployment scenario
- Identify roles of those involved
- Identify key usage scenario
- Identify technologies to be used
- Identify application security mechanisms

[1]From http://msdn.microsoft.com/en-us/library/ms978527.aspx

Sources of Information

- Documentation
- Interviewing developers
- Standards documentation
- Source code profiling
  - If source already exists
- System profiling
  - If a working version is available

2. Application Architecture Modeling

- Using information gathered, develop understanding of the proposed architecture
- To identify design concerns
- And to prioritize later efforts
- Useful to document findings using some type of model

Modeling Tools for Design Review

- Markup languages (e.g., UML)
  - Particularly diagramming features
  - Used to describe OO classes and their interactions
  - Also components and uses
- Data flow diagrams
  - Used to describe where data goes and what happens to it

3. Threat Identification

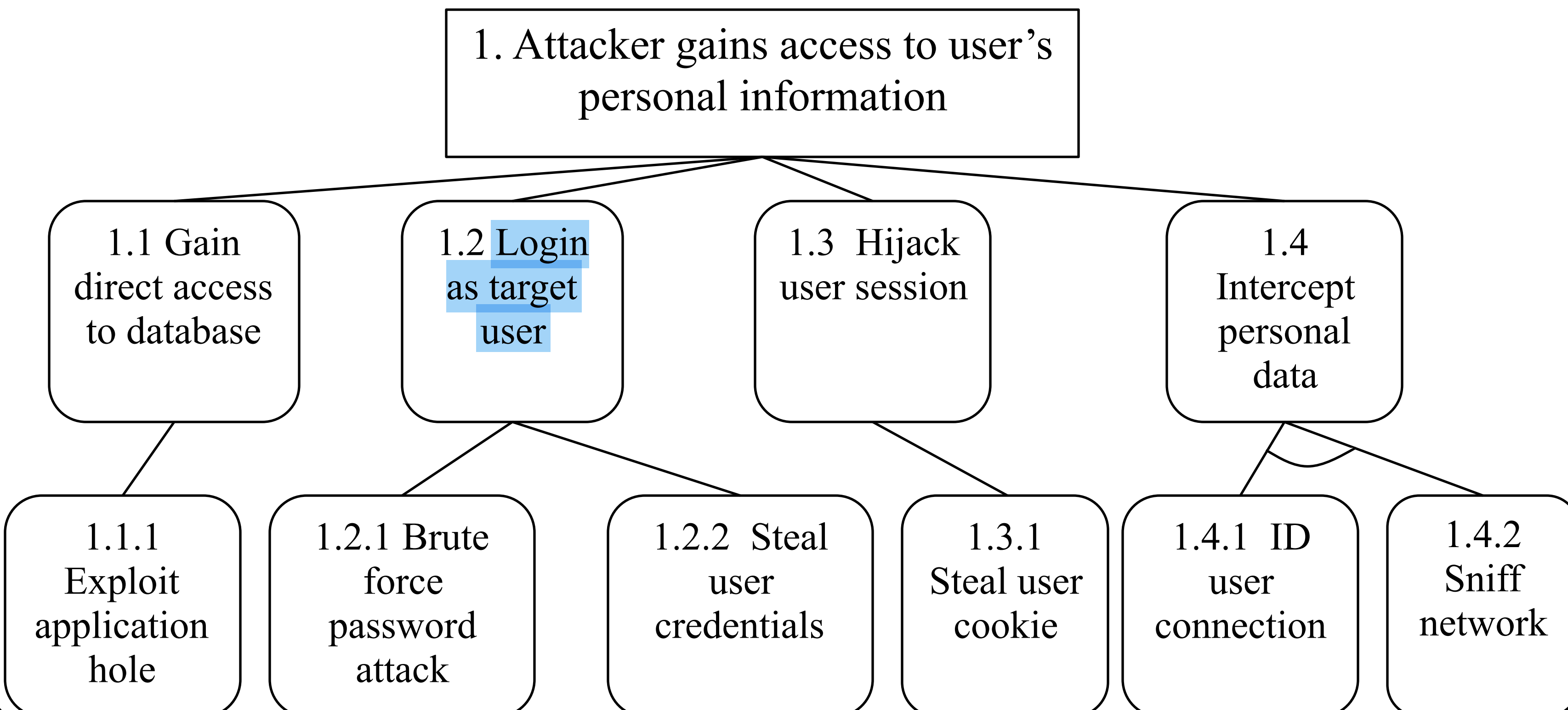
- Based on models and other information gathered
- Identify major security threats to the system's assets
- Sometimes done with *attack trees*

Attack Trees

- A way to codify and formalize possible attacks on a system
- Makes it easier to understand relative levels of threats
  - In terms of possible harm
  - And probability of occurring

A Sample Attack Tree

- For a web application involving a database
- Only one piece of the attack tree



The STRIDE Approach

- Developed and used by Microsoft
  - Part of their SDL threat modeling process[1]
- Depends on having built a good system model diagram
  - Showing components, data flows, interactions
  - Specifying where data and control cross trust boundaries
- Then, for each element, consider the STRIDE threats

[1] http://blogs.technet.com/b/security/archive/2012/08/23/microsoft-s-free-security-tools-threat-modeling.aspx

STRIDE Threats

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation Disclosure
- **D**enial of Service
- **E**scalation of Privilege

How To Apply STRIDE

- For each element in diagram, consider each possible STRIDE threat
- Some types of threats not applicable to some types of elements
- Pay particular attention to things happening across trust boundaries

4. Documentation of Findings

- Summarize threats found
  - Give recommendations on addressing each
- Generally best to prioritize threats
  - How do you determine priorities?
  - DREAD methodology is one way

DREAD Risk Ratings

- Assign number from 1-10 on these categories:
- **D**amage potential
- **R**eproducibility
- **E**xploitability
- **A**ffected users
- **D**iscoverability
- Then add the numbers up for an overall rating
- Gives better picture of important issues for each threat

5. Prioritizing Implementation Review

- Review of actual implementation once it's available
- Requires a lot of resources
- You probably can't look very closely at everything
- Need to decide where to focus limited amount of attention

One Prioritization Approach

- Make a list of the major components
- Identify which component each risk (identified earlier) belongs to
- Total the risk scores for categories
- Use the resulting numbers to prioritize

Application Review

- Reviewing a mature (possibly complete) application
- A daunting task if the system is large
- And often you know little about it
  - Maybe you performed a design review
  - Maybe you read design review docs
  - Maybe less than that
- How do you get started?

Need to Define a Process

- Don't just dive into the code
- Process should be:
  - Pragmatic
  - Flexible
  - Results oriented

I will organize how I lead the implementation review

- Will require code review
  - Which is a skill one must develop

Review Process Outline

1. Preassessment
   – Get high level view of system
2. Application review
   – Design review, code review, maybe live testing
3. Documentation and analysis
4. Remediation support
   – Help them fix the problems

- May need to iterate

Reviewing the Application

- You start off knowing little about the code
- You end up knowing a lot more
- You'll probably find the deepest problems related to logic after you understand things
- A design review gets you deeper quicker
  - So worth doing, if not already done
- The application review will be an iterative process

General Approaches To Design Reviews

- Top-down
  - Start with high level knowledge, gradually go deeper
- Bottom-up
  - Look at code details first, build model of overall system as you go
- Hybrid
  - Switch back and forth, as useful

Code Auditing Strategies

- Code comprehension (CC) strategies
  - Analyze source code to find vulnerabilities and increase understanding
- Candidate point (CP) strategies
  - Create list of potential issues and look for them in code
- Design generalization (DG) strategies
  - Flexibly build model of design to look for high and medium level flaws

Some Example Strategies

- Trace malicious input (CC)
  – Trace paths of data/control from points where attackers can inject bad stuff
- Analyze a module (CC)
  – Choose one module and understand it
- Simple lexical candidate points (CP)
  – Look for text patterns (e.g., `strcpy()`)
- Design conformity check (DG)
  – Determine how well code matches design

Guidelines for Auditing Code

- Perform flow analysis carefully within functions you examine
- Re-read code you've examined
- Desk check important algorithms
- Use test cases for important algorithms
  - Using real system or desk checking
  - Choosing inputs carefully

Useful Auditing Tools

- Source code navigators
- Debuggers
- Binary navigation tools
- Fuzz-testing tools
  - Automates testing of range of important values

    send in a bunch of test cases to see what the results are

    sampling needed because you don't have a chance to use all test values

Evaluating Running Systems

- Evaluating system security requires knowing what's going on
- Many steps are necessary for a full evaluation
- We'll concentrate on two important elements:
  - Logging and auditing

Logging

- No system's security is perfect
- Are my system's imperfections being exploited?
- You need to understand what's going on to tell
- Logging is the tool for that:
  - ***Keeping track of important system information for later examination***

The Basics of Logging

- OS and applications record messages about their activities
  - In pre-defined places in the file system
- These messages record important events
- And unexpected events
- Many attacks leave traces in the logs

Access Logs

- One example of what might be logged for security purposes
- Listing of which users accessed which objects
  - And when and for how long
- Especially important to log failures

Other Typical Logging Actions

- Logging failed login attempts
  – Can help detect intrusions or password crackers
- Logging changes in program permissions
  – A common action by intruders
- Logging scans of ports known to be dangerous

Problems With Logging

- Dealing with large volumes of data
- Separating the wheat from the chaff
  - Unless the log is very short, auditing it can be laborious
- System overheads and costs

Log Security

- If you use logs to detect intruders, smart intruders will try to attack logs
  – Concealing their traces by erasing or modifying the log entries
- Append-only access control helps a lot here
- Or logging to hard copy
- Or logging to a remote machine

Local Logging vs. Remote Logging

- Should you log just on the machine where the event occurs?
- Or log it just at a central site?
- Or both?

Local Logging

- Only gives you the local picture
- More likely to be compromised by attacker
- Must share resources with everything else machine does
- Inherently distributed
  - Which has its good points and bad points

Remote Logging

- On centralized machine or through some hierarchical arrangement
- Can give combined view of what's happening in entire installation
- Machine storing logs can be specialized for that purpose
- But what if it's down or unreachable?
- A goldmine for an attacker, if he can break in

Desirable Characteristics of a Logging Machine

- Devoted to that purpose
  - Don’t run anything else on it
- Highly secure
  - Control logins
  - Limit all other forms of access
- Reasonably well provisioned
  - Especially with disk

Network Logging

- Log information as it crosses your network
- Analyze log for various purposes
  - Security and otherwise
- Can be used to detect various problems
- Or diagnose them later

Logging and Privacy

- Anything that gets logged must be considered for privacy
- Am I logging private information?
- If so, is the log an alternate way to access it?
- If so, is the log copy as well protected as the real copy?

An Example

- Network logs usually don't keep payload
  - Only some header information
- You can tell who talked to whom
- And what protocol they used
- And how long and much they talked
- But not what they said

Auditing

- Security mechanisms are great
  – If you have proper policies to use them
- Security policies are great
  – If you follow them
- For practical systems, proper policies and consistent use are a major security problem

Auditing

- A formal (or semi-formal) process of verifying system security
- “You may not do what I expect, but you will do what I inspect.”
- A requirement if you really want your systems to run securely

Auditing Requirements

- Knowledge
  - Of the installation and general security issues
- Independence
- Trustworthiness
- Ideally, big organizations should have their own auditors

When Should You Audit?

- Periodically
- Shortly after making major system changes
  - Especially those with security implications
- When problems arise
  - Internally or externally

Auditing and Logs

- Logs are a major audit tool
- Some examination can be done automatically
- But part of the purpose is to detect things that automatic methods miss
  – So some logs should be audited by hand

What Does an Audit Cover?

- Conformance to policy
- Review of control structures
- Examination of audit trail (logs)
- User awareness of security
- Physical controls
- Software licensing and intellectual property issues

Does Auditing Really Occur?

- To some extent, yes
- 2008 CSI/FBI report says more than 64% of responding organizations did audits
- Doesn't say much about the quality of the audits
- It's easy to do a bad audit

Conclusion

- Don't assume your security is perfect
- Either at design time or run time
- Using security evaluation tools can help improve your security
- Necessary at all points in the life cycle:
  - From earliest design until the system stops operating