Secure Design

Computer Security I

CS461/ECE422

Fall 2009

Reading Material

- Chapter 19 of Computer Security: Art and Science
- Threat Modeling by Frank Swiderski and Window Snyder
- Build Security in Portal https://buildsecurityin.us-cert.gov/
  - Particularly Article on Risk-Base and Functional Security Testing
  - https://buildsecurityin.us-cert.gov/daisy/bsi/255-BSI.htr

Outline

- Secure Design
  - Best Practices
  - Security Requirements
  - Assurance Techniques
- Threat Modelling
- Other Design/Development Issues
- Testing

Goals for Secure Development

- Correct Operation
  - System does what it supposed to do
- Secure Operation
  - System operation cannot be corrupted
- Assured System
  - Evidence that system operates within specified security and feature requirements

Secure Design

- Good software engineering principles
  - Common sense
  - Stuff you know you should be doing
  - An art not a science. Valuable to review and be aware of
- Presence of bugs in general provide opportunity for security vulnerabilities
- Security addressed up front
  - Built in vs retro-fit

Best Practices

- Discussed 8 design principles
- Numerous other Check Lists and best Practices documents
  - GASSP http://www.auerbach-publications.com/dynamic_data/2334_1221_
  - http://csrc.nist.gov/pcig/
  - Security at a Glance Checklist http://www.securecoding.org/companion/checklists/SAG/
- Check lists are useful, but should not be followed blindly
  - Dependent on application domain, organization, technology
- Newer tools integrate best practice enforcement
  - E.g. Numega, Rational

Security Architecture

- High level design that addresses the security requirements
- Model that lets the designers and developers reason about the security functions of the system
  - Metaphors for security can be useful
    - E.g. think about folders and filing cabinets in sheds
- Same security architecture can be reused between similar applications
  - E.g., can use same style of security architecture over multiple client-server applications

Layered Architecture

- Can address security at any or all layers
  - Application
  - Service/Middleware
  - Operating system
  - Hardware

Security Requirements

- Security is generally non-functional
  - e.g., Application should be secure against intruders
- Need to make requirements more precise
  - Version 1: “Users must be identified and authenticated”
  - Version 2: “Uses of system must be identified and authenticated by system”
  - Version 3: Adds “... before system performs any actions on behalf of user”
- Ideally can map to existing precise requirements

Ways to identify security requirements

1. Extract requirements from existing standards like Common Criteria
2. Combine threat analysis with existing policies
3. Map to existing model like BLP

Security Requirement Completeness

- Justify security requirements by associating requirements with threats
- Identified during project requirements phase
  - Use security requirements to drive security architecture
  - Identify assets to protect
    - Rank importance of asset
    - Cost/benefit

Example Threat

- Threat T1: Person not authorized to use the system gains access by impersonating authorized user
- Requirement IA1: User session must begin with proof of authentication
- Assumption A1: The product must be configured such that only the approved group of users has physical access to the system
- Assumption A4: Passwords generated by admin will be distributed in secure manner

Design Documents

- Security Functions
  - High level function descriptions
  - Mapping to requirements
- External Interfaces
  - Functional specification
- Internal Design Description for each component
  - Overview of parent component
  - Detailed description
  - Security relevance
- Literate programming tools can help with Interface and Internal Docs
  - e.g., Java doc and Doxygen

Means of Assurance

- Requirements tracing
  - Mapping security requirement to lower design levels
  - Map security design elements to implementation
  - Map security implementation to test
- Informal Correspondence
  - Ensure specification is consistent with adjacent levels of specification

Other Design Assurance Options

- Informal Arguments
- Formal Methods
  - Theorem provers
  - Model Checkers
  - UML to some degree
    - UML tools can drive this formalism down to implementation and test
- Review Meetings

Threat Modeling

- Similar to risk analysis
  - Discussed in *Threat Modeling* by Frank Swiderski and Window Snyder
  - Also UML notation
  - http://coras.sourceforge.net/index.html
- Systematically analyze code
  - Entry points, use scenarios, data flow diagrams
  - Number everything
- Develop threat models or attack trees
  - Use to drive necessary mitigations/counter measures

Adversary's Point of View

- Analyze entry points
  - Where the attacks must start
  - Uniquely number entry points
- Understand assets
  - What is goal of attack
- Trust levels
  - Expected privilege levels associated with each entry point

Entry Point Analysis

- For each entry point document
  - Name, id, description, trust levels
- Example, web listening port
  - Id = 1
  - Description = The port that the web server listens on.
  - Trust Levels
    - 1 – remote anonymous user
    - 2 – remote user with login credentials
    - 3 – Insurance Agent
    - 4 – Web admin

Characterize System Security

- Use Scenarios
  - Document how the system is expected to be used
  - E.g., web server will communicate with database on private network
- Identify assumptions and dependencies
  - E.g. web server depends on security of underlying session management

Data Flow Diagrams

- Models
  - Where entry points are used
  - external entities
  - changes of protection domain
- DFD’s can be nested

Example DFD

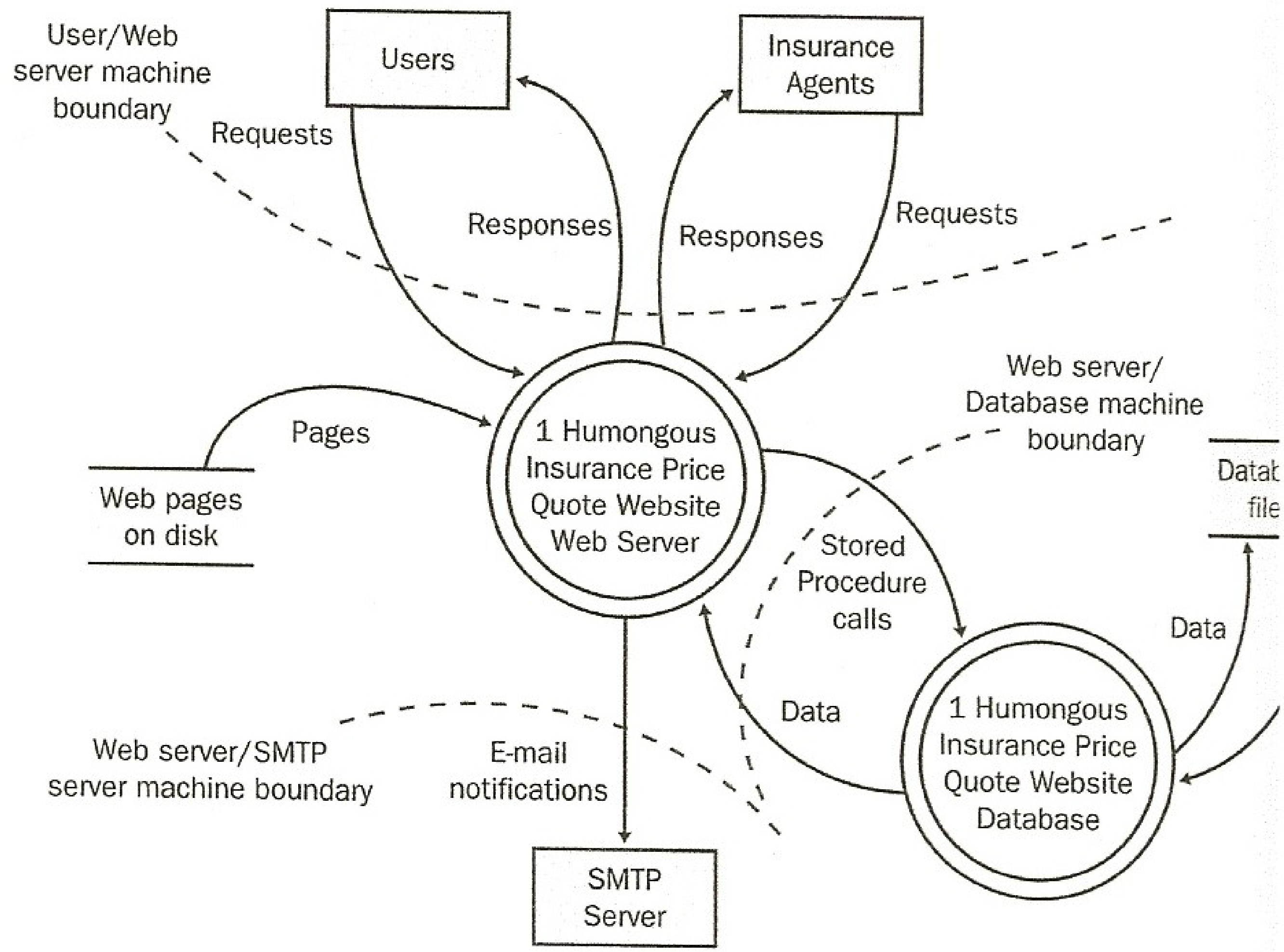


Figure 4-17 Context diagram for the Humongous InsurancePrice Quote Website.

Threat Profiling

- Start by looking at the assets
- STRIDE classification
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of privilege

Example Threat Profile

- ID = 1
- Name = adversary supplies malicious data in a request targeting the SQL command parsing engine to change execution
- STRIDE = tampering, elevation of privilege
- Mitigated? = no
- Entry points = (1.1) Login page, (1.2) data entry page, (1.3) Insurance agent quote page
- Assets = (16.3) Access to backend database

Threat Tree

- Also called attack trees
- Break a threat into underlying conditions
- Analyze paths in tree
  - If at least one step in each path is mitigated (counter-measure applied) threat is mitigated
- DREAD
  - Damage Potential
  - Reproducibility
  - Exploitability
  - Affected User
  - Discoverability

Example Threat Tree

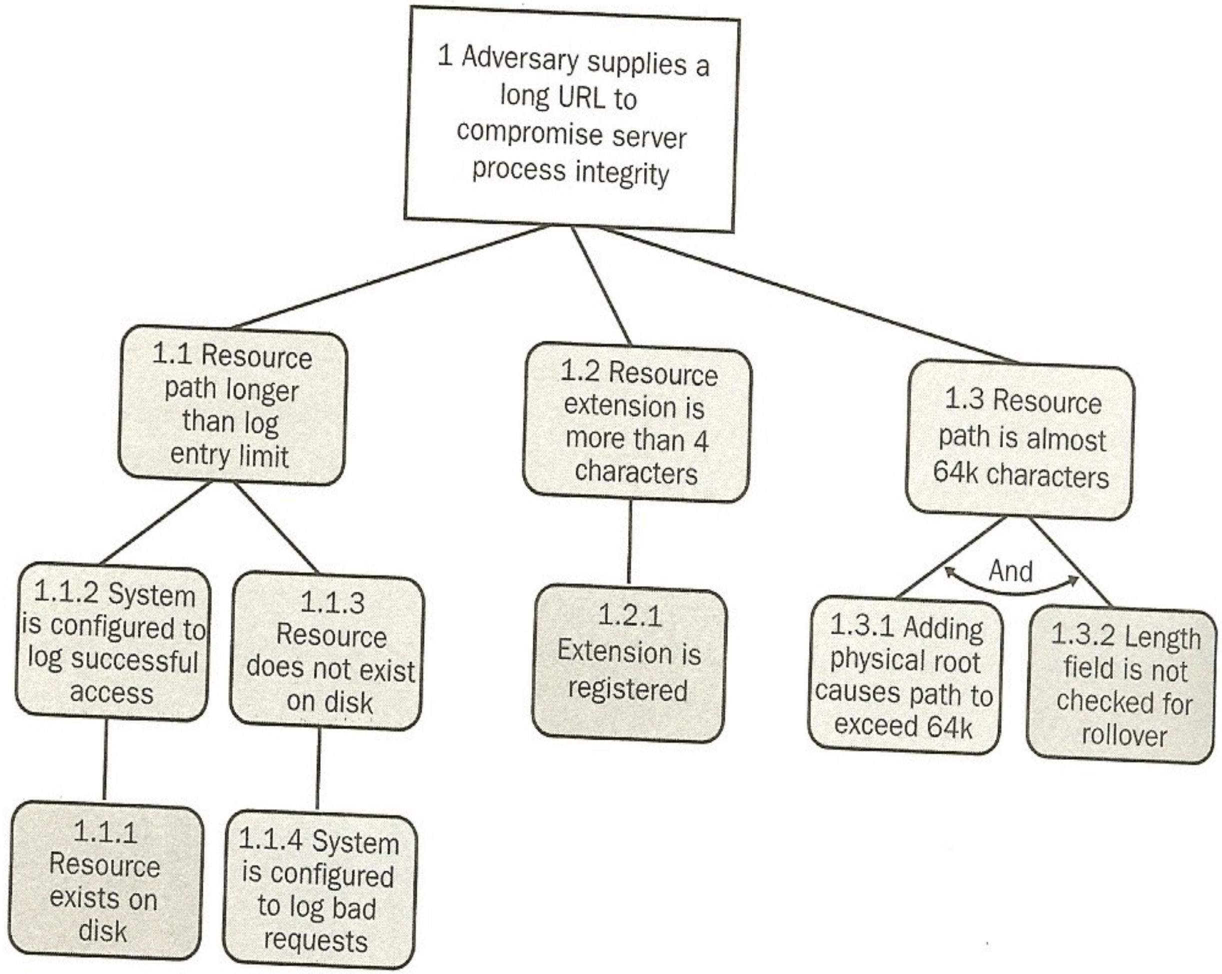


Figure 5-3 A threat tree for an adversary supplying a long URL to a Web server.

Another Example Threat Tree

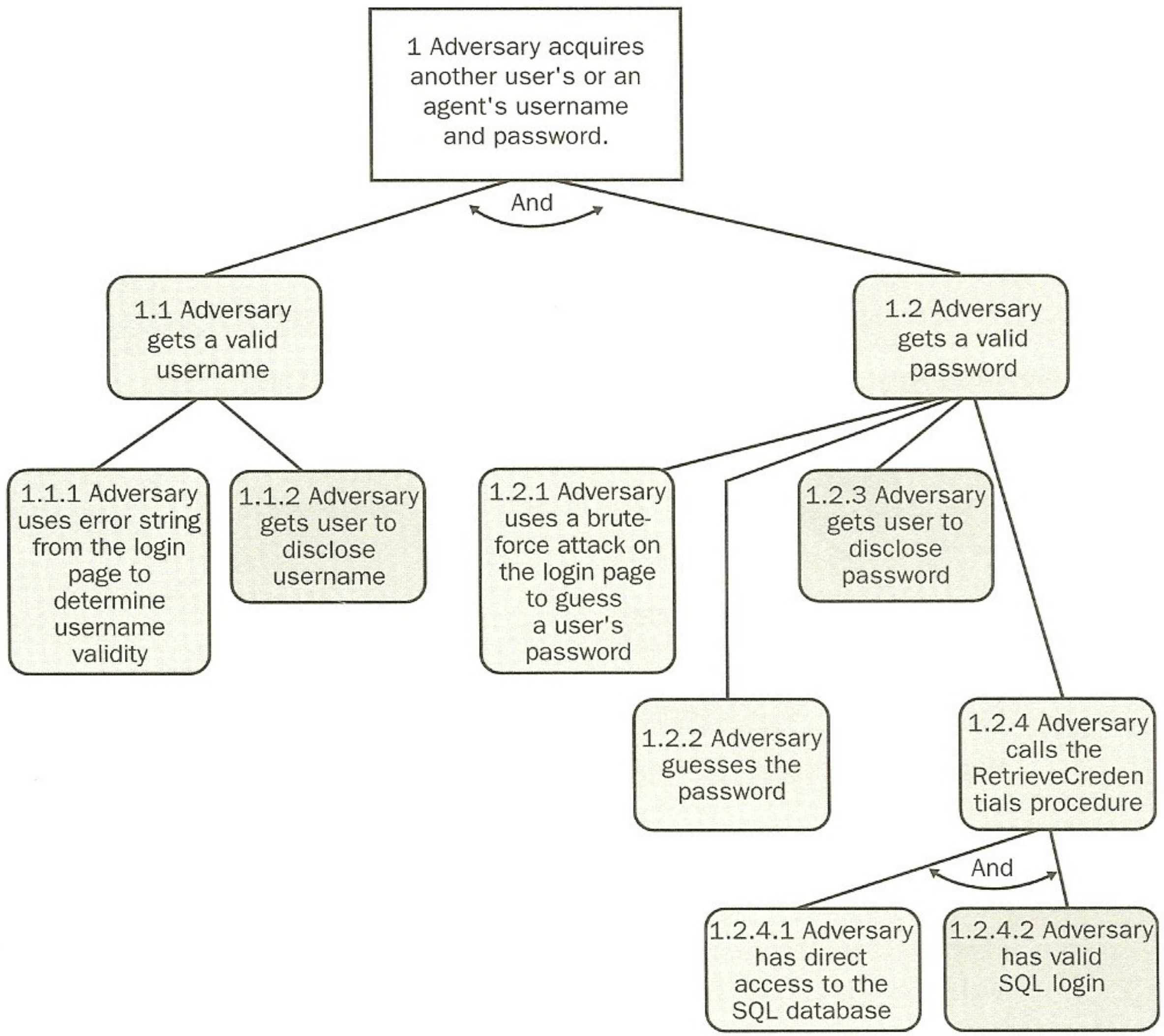


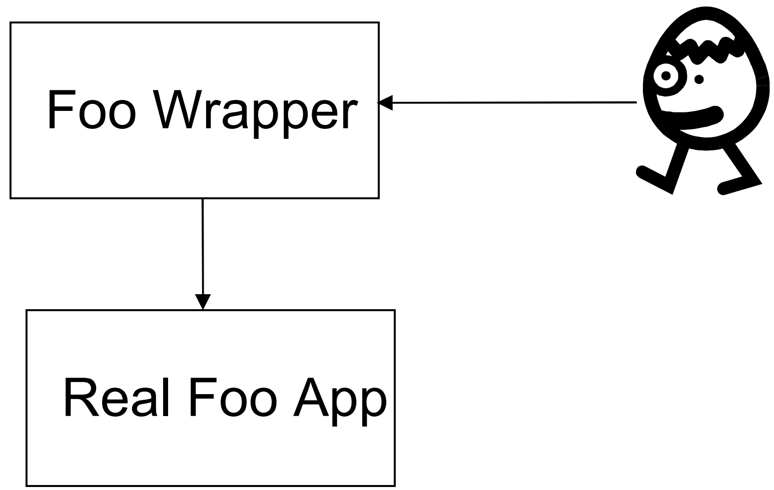
Figure 5-5 Threat tree for Humongous Insurance Price Quote Website.

Retrofit Design

- Wrapper approach
  - Write program to cleanse input before sending it to the “real” program. Similarly cleanse output before return
- Interpose approach
  - Write another program to sit between caller and original program. Much like firewall proxies
- Isolate
  - Chroot and Java jails. Create an environment where the ill-behaving program cannot cause too much harm

Wrapper Example

- Wrapper cleans input and environment
- Invokes real app on cleansed input in restricted environment



Design Separation Options

- Frequently it is desirable to minimize/control communication between different parts of the system
  - Physical separation
  - Temporal separation
  - Cryptographic Separation
  - Logical separation
    - relying on reference monitor
    - E.g. Separate processes
  - Virtualization
    - Create multiple copies of the OS
    - E.g. VM Ware

Configuration Management

- Control committed changes to the system
- Version control and tracking
  - Be able to recreate version 1.2.3.68
- Change authorization
  - All committed changes must be entered by team leader during final stages of development
  - Team member can only commit approved files
- Integration procedures
- Tools for product generation

Security Testing

- Look at the problem in a non-standard way. Or work with others who can.
  - E.g., using privileged mouse driver to co-opt system
  - Standard issue of not being good testers of our own code
- Designing for testing
  - Well defined API's and documentation to enable good test design

Many kinds of testing

- Unit testing
  - Use integrated tools like JTest
- Functional Testing (Black box)
  - Test based on feature requirements
- Code based or structural testing (White box)
- Ad Hoc/Exploratory Testing
- Boundary Value Analysis

Special Problems of Security Testing

- Different motivations for finding bugs in the field
  - Malicious intent
- Often negative testing
  - Testing for absence of item
  - E.g., unauthorized users should not be able to access account data
- Security requirements are often vague
- Requires thinking at different levels of abstraction
  - E.g., must understand the guts of strcpy to know that it can be exploited
- Looking at completeness rather than the common case

Risk-based Testing

- Use Threat Models/Attack trees to drive test cases
- Order tests by highest risk
  - Never have enough time to test all possible combinations

Test Coverage

- Particularly important to ensure that error handling cases are tested
  - Frequently not exercised and source of lurking errors
  - Tools exists to track test coverage

Key Points

- Security requirements driven by threats
  - Requirements drive architecture
  - Threat modeling drives design and testing
- Security testing has unique difficulties
  - Negative Testing
  - Thinking outside the box