Threat Modeling

SecAppDev 2010
Design flaws are major

Difficult and costly to fix post facto
  – Too costly to fix?
Often made because of false assumptions
  – Trust models
  – Underestimate attackers
  – Naïveté
  – Ignorance
Let’s look at design activities

Approaches
– Risk analyses
– Threat modeling

Formal vs. ad hoc
– Same intent

Find problems before we implement them
Let’s see what the SDLCs offer

Several to choose from

Enough good in each to consider all

– Look carefully at each author’s perspective

Apply consistently and measure
Who are the players?

Microsoft
– Secure Development Lifecycle

Cigital
– “Touchpoint” process

OWASP
– Comprehensive Lightweight Application Security Process (CLASP)
SDL: Product Risk Assessment

Analyze the product’s functions and their “danger” levels
  – Use their sample questionnaire as a starting point
Determine the privacy impact
How much effort should be applied?
SDL: Risk Analysis

This one really comes down to
- Threat modeling
- Using threat model to aid code review
- Using threat model to aid testing
- Determine key success factors and metrics

Guided by
- STRIDE (Spoofing, Tampering, Repudiation, Info disclosure, DoS, Elevation)
- DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability)
Cigital’s “Touchpoints”

Built by McGraw et al over time
- Perspective is consulting services

Consists of three pillars
- Risk management
- Knowledge
- Touchpoints
The Touchpoints
A risk management framework

Business goals determine risks
Risks drive methods
Methods yield measurement
Measurement drives decision support
Decision support drives fix/rework and application quality
Architectural risk analysis

Build a one page white board design model

Use hypothesis testing to categorize risks
– Threat modeling/Attack patterns

Rank risks

Tie to business context

Suggest fixes

Repeat
Architectural risk analysis

Follow a process
Build an overview (one page)

Three steps
– Attack resistance analysis
– Ambiguity analysis
– Weakness analysis

Rank risks
Build mitigations
Attack resistance

Identify general flaws
- Non-compliance
- Where guidelines are not followed

Map applicable attack patterns

Identify risks in architecture

Consider known attacks against similar technologies

Attack Patterns
- Pattern language
- Database of patterns
- Actual flaws from clients

Exploit Graphs
- Ease mitigation
- Demonstrate attack paths

Secure design
Knowledge: Attack patterns

Attack Pattern Schema

Description

General Indication

Recipes for exploit

Protection schemes

Indications

– Signs of weakness
– Specific concrete properties of the software
– Easily detectable

Protection Schemes

Guard Component

"Protected" Interface

Attack Target
a.k.a The Jewels
Knowledge: 48 attack patterns

Make the Client Invisible
Target Programs That Write to Privileged OS Resources
Use a User-Supplied Configuration File to Run
Commands That Elevate Privilege
Make Use of Configuration File Search Paths
Direct Access to Executable Files
Embedding Scripts within Scripts
Leverage Executable Code in Nonexecutable Files
Argument Injection
Command Delimiters
Multiple Parsers and Double Escapes
User-Supplied Variable Passed to File System Calls
Postfix NULL Terminator
Postfix, Null Terminate, and Backslash
Relative Path Traversal
Client-Controlled Environment Variables
User-Supplied Global Variables (DEBUG=1, PHP Globals, and So Forth)
Session ID, Resource ID, and Blind Trust
Analog In-Band Switching Signals (aka “Blue Boxing”)
Attack Pattern Fragment: Manipulating Terminal Devices
Simple Script Injection
Embedding Script in Nonscript Elements
XSS in HTTP Headers
HTTP Query Strings
User-Controlled Filename
Passing Local Filenames to Functions That Expect a URL
Meta-characters in E-mail Header
File System Function Injection, Content Based
Client-side Injection, Buffer Overflow
Cause Web Server Misclassification
Alternate Encoding the Leading Ghost Characters
Using Slashes in Alternate Encoding
Using Escaped Slashes in Alternate Encoding
Unicode Encoding
UTF-8 Encoding
URL Encoding
Alternative IP Addresses
Slashes and URL Encoding Combined
Web Logs
Overflow Binary Resource File
Overflow Variables and Tags
Ambiguity analysis

Consider implications of design
Generate separate arch. diagrams
Unify understanding
  – Uncover ambiguity
  – Identify downstream difficulty (traceability)
  – Unravel convolution

Apprenticeship model
Use system, technology experts
  – Win32 knowledge
  – JVM/managed code
  – Language/compiler knowledge
Previous experience
Weakness analysis

Consider systemic flaws
- COTS
- Frameworks
- Network topology
- Platform

Identify services

Map weaknesses to assumptions

Experience base
- Assessments of COTS and platforms

Attack patterns

Other resources
- Mailing lists
- Product documentation
Enter threat modeling

We seek to enumerate
- Who
- What
- How
- Impact
- Mitigation

Order does not matter

Spreadsheets can help
Who is the threat agent?

Who has access?
What are their motivations?
How resourceful are they?
How knowledgeable?
What is the attack target?

What is the target of the attack?
– From high level to low level

Start with asset inventory
– Consider business value
– Technical significance
– Security dependencies
How will the attack work?

This is the biggie
What kinds of attacks are relevant?
– Available tools to automate
– Manual attacks
– Outside vs. inside
– Authenticated vs. not
– Remote vs. local
What is the impact?

What is the business impact of an attack?

– Direct costs
– Indirect costs
– Reputation impact
– Down time
How can it be mitigated?

How can we reduce likelihood of each attack?
- Mechanism
- Costs
- Feasibility
- User acceptance
Start with a diagram

Top level to visualize functionality
– Think business, not technology
– NOT a network diagram

Identify the assets
Annotate with design patterns
– Servlet, thin client, session, MVC, etc.
Attack surface

Enumerate the entire attack surface
– Top to bottom
What interfaces are available, and how?
– APIs
– UIs
– B2B
– OS
– Signals
– Etc
Security zones

Break down architecture into separate zones
- Client (PC) zone
- Middleware zone
- Application zone
- Database zone

Enumerate all
- Components
- Agents
Enumerate attacks

Attack trees help
- Start with high value outcome and reverse engineer
  - Dive into fine level of detail
  - List preconditions for each step

It is OK to fail
Some attacks may seem extreme and unlikely
Consider abuse/misuse case attacks too
Include all components

Key security components
- Authenticators
- Access controllers
- Input validators
- Output encoders

Identify security controls explicitly
Consider data flow

Identify and trace key data assets
– Credentials
– Account data
– Customer PII
– Crypto keys
Let’s try a simple one

Company background
- Global telecommunications company, USD$20B/yr
- Carrier grade reliability
- “Brick and mortar”
- Very little business on net

Data processing in two groups
- Production data
- Administrative
  - Provisioning
  - Accounting/billing

Provisioning done via legacy mainframe
- MVS/DB2 monster
Your project requirements

Mobile app for provisioning
  – Connects to DB2 back end

Functional reqs
  – Field techs securely access provisioning eng
  – View/alter customer acct info

– All services of DB2 user
– Only authorized techs may use
– If device lost, no customer data lost
– If net not avail, app must cache commands and execute when avail
Business risks

What are the top business risks?
What are the top attack targets?
What abuse cases are likely?
Who are the threat agents?
Design

Celltop
- Java app
- Renderer
- Authenticator

Mobile prov engine
- Authenticator

Prov engine
## Threat analysis

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>How</th>
<th>Impact</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What have we learned?

What glaring problems surfaced through this analysis?
Considerations in Choosing

One size does NOT fit all

Cultural issues
  – Dev org size
  – How “process heavy” are you now?
  – Across entire organization
Plan Your Own Hybrid

Look at each process
Which components are likely to work best for you?
– Feasibility is vital
– Sometimes best isn’t better

Think things through carefully
Plan of Action

What is in place now?
Target process
Gap analysis
Chart a course
  – Small steps
  – Defect data helps to prioritize steps
Buy-in is essential