Input validation: the Good, the Bad and the Ugly

Johan Peeters
independent
http://secappdev.org
http://johanpeeters.com
Motivation

Applications evolve
Aim

Validation architecture
Case study

- CRUD securities
- Forms for
  - Create
  - Search
  - Update
- List search results
  - View details
  - Open for modification
  - Delete
Case study

- Web server
  - presentation
- Application server
  - business logic
- RDBMS
  - data

controller → model → view
JavaScript in the web browser

- JSPs with script tags
- Custom-built tag library
  - Populate drop-down boxes
  - Standard event handlers
- JavaScript

```html
<html>
<head>
  <script src="typeCheck.js"/>
</head>
<body>
  <form action="add" method="post">
    <input type="submit" value="Add"/>
    <select>
      <option value="bond">Bond</option>
      <option value="share">Share</option>
    </select>
    <input type="text" onkeypress="return isNumeric(event)"
           onchange="return checkSum(event)"/>
  </form>
</body>
</html>
```
Struts on the web tier

controller

ActionServlet

doPost
doGet
...

RequestProcessor

process
...

Model

Action
execute
...

ActionForm

validate
...

Controller

view:
.jsp

RequestDispatcher
forward
include
...

request, response

request, response

mapping, form, request, response

mapping, request
EJBs on the business tier

SecurityDTO
- int previousHash
- <type> <property>
- get<property>
- set<property>
- calculateHash
...

SecurityFacadeBean

validate
store
...

OWASP
RDBMS as data tier

- Not NULL
- Referential constraints
- Columns are typed
- Unicity constraints

preparedStatement

RDBMS
Appraisal

- Validation does what it is supposed to
- Validation code is ugly
- Duplication leads to entropy
  - Maintenance nightmare
Defense in depth

[Diagram with layers: User, Web server, Application server, RDBMS]
Back to basics

- Integrity
- Validation serves to
  - Provide good user experience
  - Protect against malicious users
Conjecture

- Validate to enhance user experience near user
- Validate to protect against malicious users near data
The front-end revisited

- Provide great user experience
- Embrace JavaScript
- Embrace AJAX
The back-end revisited

- Defend against malicious use
- Supplement data types, not NULL, unicity and referential integrity with
  - Check constraints
  - Triggers
    - Per statement
      - Before
      - After
    - Per row
      - Before
      - After
  - Stored procedures
Challenges

- Error handling
  - No big deal: no mercy if you bypass client
- Duplication of business logic
Rooting out duplication

- If single formalism is used, perhaps the same code can be called from the different tiers
- Expressing the same integrity constraints in different formalisms can bring advantages
- Single source distributed over several tiers
Examples of single source to multiple tiers

- Ruby on Rails
- GWT
- Swift
Ruby on Rails code generation

- MVC
- No business tier
- Model is an ActiveRecord instance, its class
  - Contains the business logic
  - maps to a table in RDBMS
  - generated by a model generator
- View is a template with Ruby code
- RDBMS end
  - Model generator also generates DDL
  - Validation in the model, not in the RDBMS
- JavaScript generated by helpers
  - Prototype
  - Script.aculo.us
GWT compiles Java code to JavaScript

- Swing-like APIs for building web user interfaces
- Code annotated as ‘client’
  - Is compiled to JavaScript
  - Executes in the browser
  - E.g.
    - com.johanpeeters.gwt.experiment.client
    - com.johanpeeters.gwt.experiment.server
- Shared client- and server-side validation code is possible
Swift places code on client or server according to security requirements

- Annotate variable declarations with security constraints
  - `int {server -> server; server <- server} secret`
  - `int {server -> client; server <- server} tries`

- Partitions application into
  - Client
  - Server

- Compiles to client and server GWT packages
- GWT compiles client packages to JavaScript
Questions and Comments
Thank you!

yo ‘at’ johanpeeters.com
yo ‘at’ secappdev.org