

Verifying Security of Web Applications

Marius Minea

joint work with Petru Florin Mihancea

Politehnica University of Timișoara
Romania



Clarke Symposium, CMU
September 19, 2014

Verifying security is important

Model checking can be done for:

- hand-written models

- binaries of actual implementations

- models extracted from applications* (source/byte-code)

This talk: extracting models from JSP web applications

Extracting Models from Web Applications

Tool: *jModex* [Mihancea & M., CSMR-WCRE'14]
tailored to applications written using JavaServer Pages

input: user request parameters, e.g., {(name, pepi), (age, 33)}
must handle *sets* of pairs (strings)

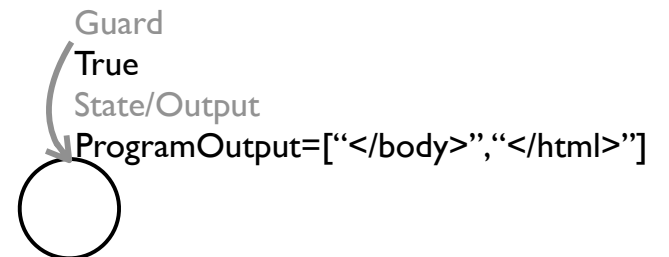
output: generated HTML

state: session attributes, often databases

⇒ translate JSP code into extended finite state machine (EFSM)

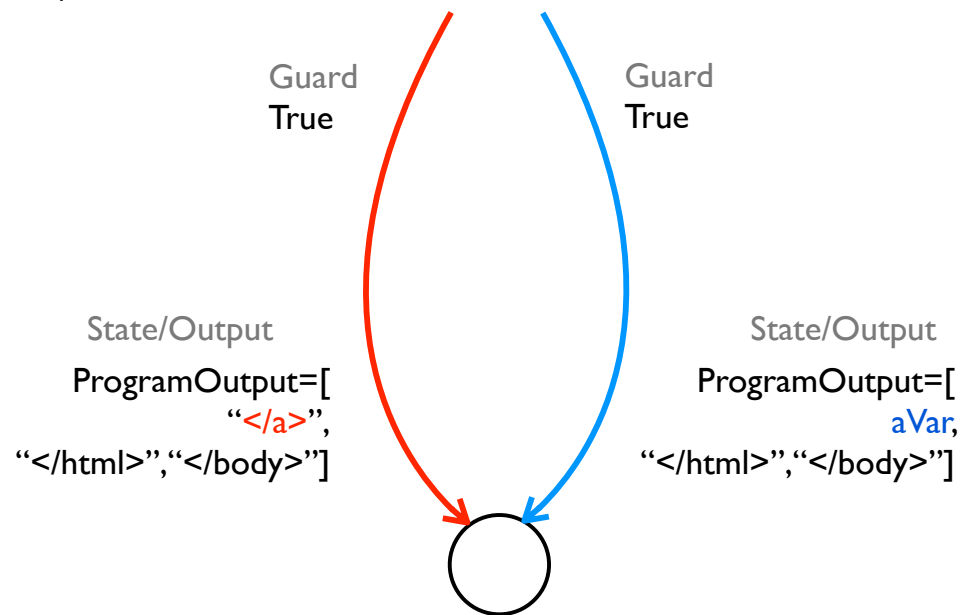
EFSM Building

```
void _jspService(...) { // A component endpoint
    String aVar = "No Link!";
    out.println("<html>");
    out.println("<body>");
    if(request.getParameter("update").equals("false")) {
        out.println(aVar);
    } else {
        request.getSession().setAttribute("seen", "true");
        out.println("<a href=\"B.jsp?name=exec\"");
        out.println("Click here!");
        out.println("</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```



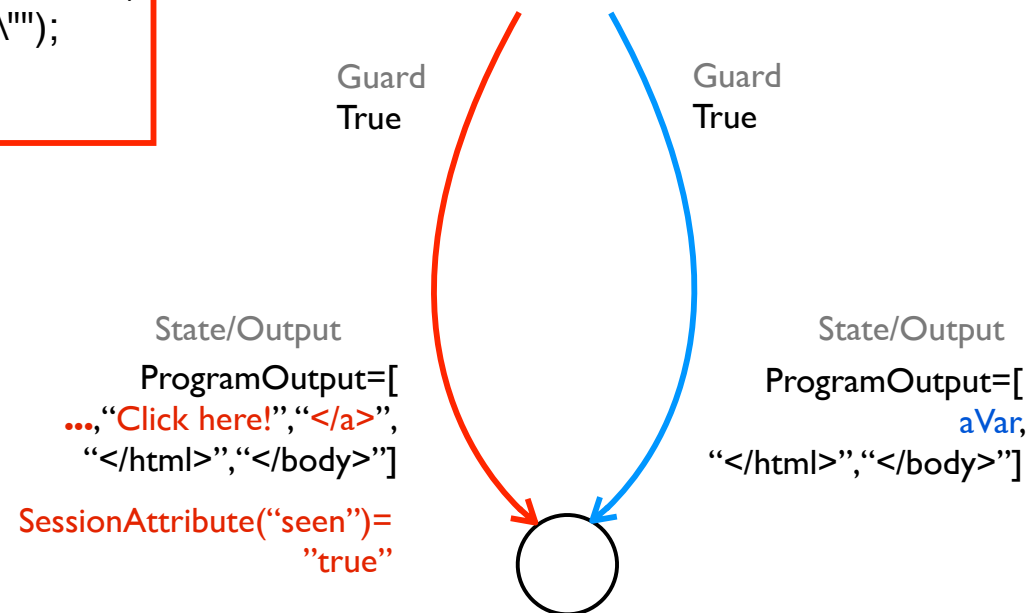
EFSM Building

```
void _jspService(...) { // A component entrypoint
    String aVar = "No Link!";
    out.println("<html>");
    out.println("<body>");
    if(request.getParameter("update").equals("false")) {
        out.println(aVar);
    } else {
        request.getSession().setAttribute("seen", "true");
        out.println("<a href=\"B.jsp?name=exec\">");
        out.println("Click here!");
        out.println("</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```



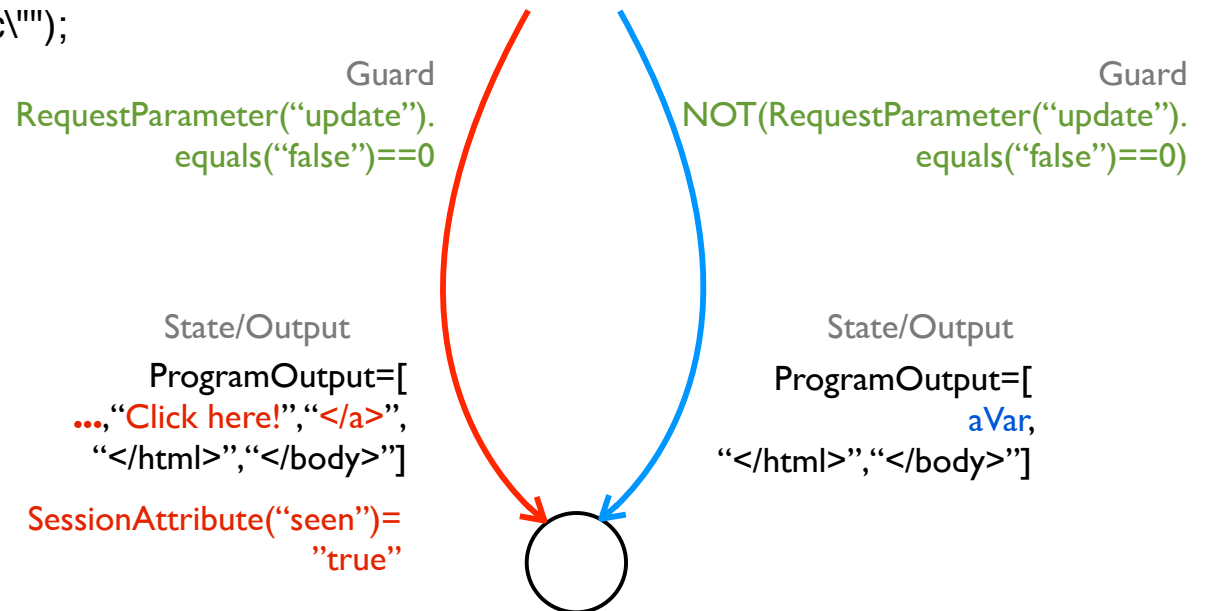
EFSM Building

```
void _jspService(...) { // A component endpoint
    String aVar = "No Link!";
    out.println("<html>");
    out.println("<body>");
    if(request.getParameter("update").equals("false")) {
        out.println(aVar);
    } else {
        request.getSession().setAttribute("seen", "true");
        out.println("<a href='\"B.jsp?name=exec\"'>");
        out.println("Click here!");
        out.println("</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```



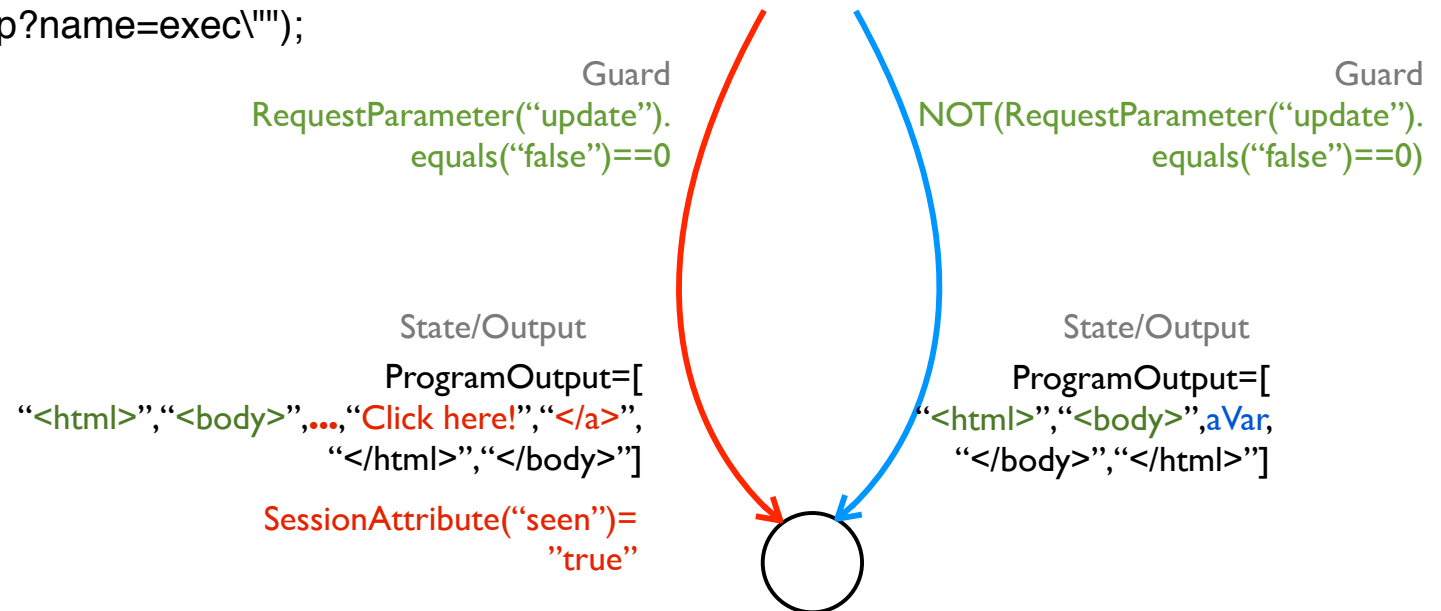
EFSM Building

```
void _jspService(...) { // A component endpoint
    String aVar = "No Link!";
    out.println("<html>");
    out.println("<body>");
    if(request.getParameter("update").equals("false")) {
        out.println(aVar);
    } else {
        request.getSession().setAttribute("seen", "true");
        out.println("<a href='\"B.jsp?name=exec'\">");
        out.println("Click here!");
        out.println("</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```



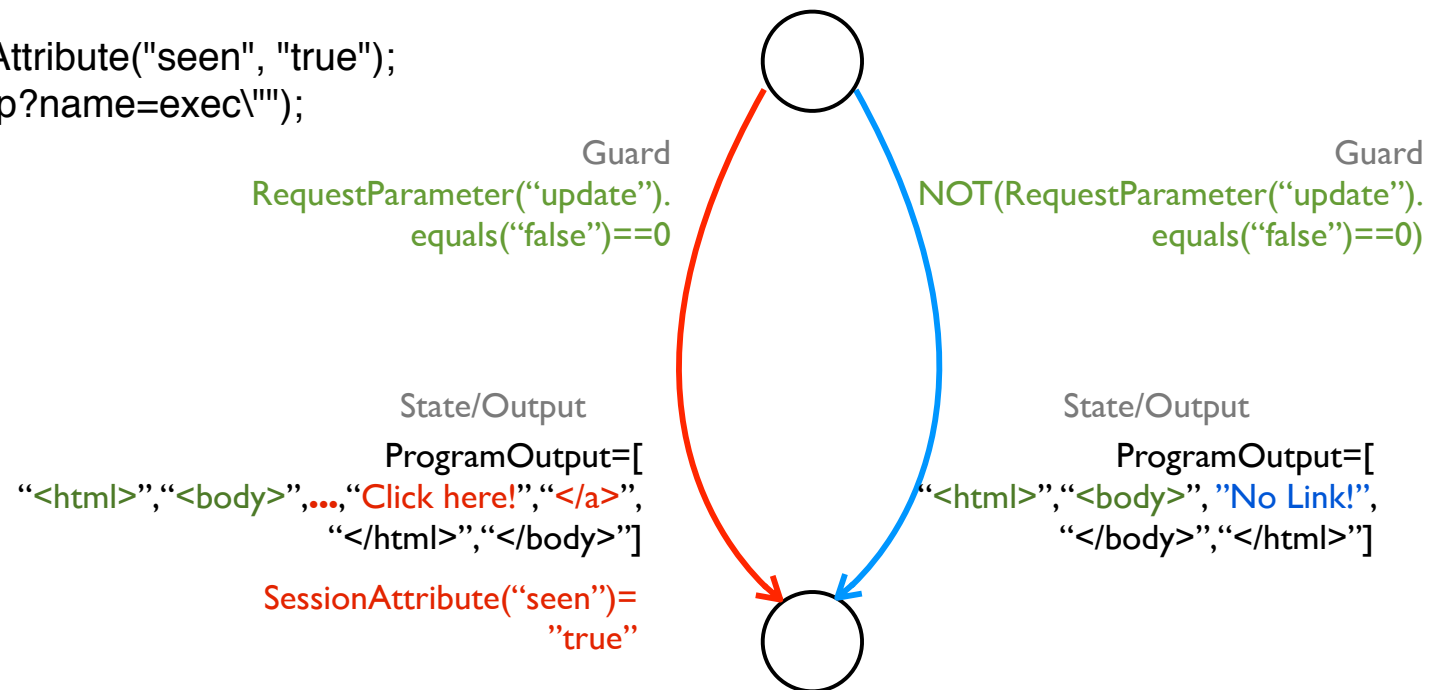
EFSM Building

```
void _jspService(...) { // A component endpoint
    String aVar = "No Link!";
    out.println("<html>");
    out.println("<body>");
    if(request.getParameter("update").equals("false")) {
        out.println(aVar);
    } else {
        request.getSession().setAttribute("seen", "true");
        out.println("<a href='\"B.jsp?name=exec'\">");
        out.println("Click here!");
        out.println("</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```



EFSM Building

```
void jspService(...) { // A component endpoint
  String aVar = "No Link!";
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
    out.println(aVar);
  } else {
    request.getSession().setAttribute("seen", "true");
    out.println("<a href='\"B.jsp?name=exec'\">");
    out.println("Click here!");
    out.println("</a>");
  }
  out.println("</body>");
  out.println("</html>");
}
```

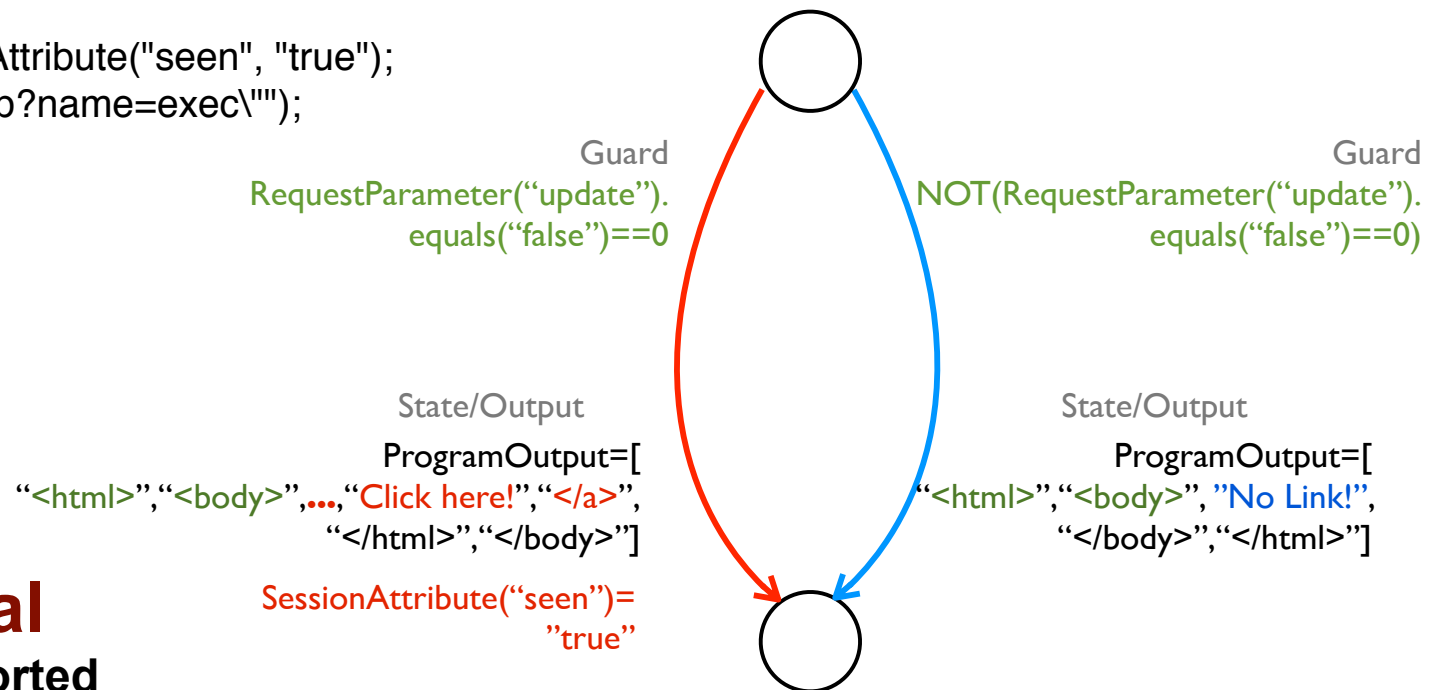


EFSM Building

```
void _jspService(...) { // A component entrypoint
  String aVar = "No Link!";
  out.println("<html>");
  out.println("<body>");
  if(request.getParameter("update").equals("false")) {
    out.println(aVar);
  } else {
    request.getSession().setAttribute("seen", "true");
    out.println("<a href='\"B.jsp?name=exec'\">");
    out.println("Click here!");
    out.println("</a>");
  }
  out.println("</body>");
  out.println("</html>");
}
```

Inter-procedural
recursion not supported

Loops
state node for each loop header



From EFSM to ASLan++ and Model Checking

jModex designed for specific language and model checkers
part of the **SPaCIoS** FP7 project www.spacios.eu
(Secure Provision and Consumption in the Internet of Services)

ASLan: low-level language with semantics based on term-rewriting
incorporating Dolev-Yao intruder model

ASLan++: higher-level language, closer to usual imperative syntax

Model checkers:

CL-AtSe (Loria-INRIA), constraint logic

OFMC (IBM/ETHZ), on-the-fly

SATMC (U. Genova), sat-based

Model extraction in a nutshell

jModex

user request parameters

e.g. {(name,pepi), (age,33)}

Application



servlet

**“global” state like
database or session
attributes**

e.g. {(userid,5)}

**returns
an HTML**

```
void _jspService(.....) { //An entry point
    if(request.getParameter("update").equals("true"))
        request.getSession().setAttribute("seen", "T");
    else
        request.getSession().setAttribute("seen", "F");
}
```



```
body {
while(true) { //the server loop
select {
on (?RU*->*Actor:rEntry(?RParams)): { //Component selection
//and the set of params
select {
on (RParams->contains((supdate, true))): { //PATH1
if(RSess->contains((sseen, ?Sseen))) {
RSess->remove((sseen, Sseen));
}
RSess->contains((sseen, sT));
}
}
on (RParams->contains((supdate, ?Supdate)) &
!(RParams->contains((supdate, true)))): { //PATH2
if(RSess->contains((sseen, ?Sseen))) {
RSess->remove((sseen, Sseen));
}
RSess->contains((sseen, sF));
}
}
}
}
}}}
```

Special jModex features

Model *database operations*

needed since many applications use SQL
supports essential subset (select, update, delete, etc.)

User-specified *abstractions*

e.g. functions that can be ignored (identity)
critical to reduce model size, dependent on analysis goal

Extensible

not hard-coded for JSP, adaptable to other technologies

Case Study: GotoCode Bookstore

open-source, 28 JSPs, 22 kLOC Java code, several security checks

The screenshot shows a web browser window at <http://localhost:8080/bookstore/>. The page features a navigation menu with icons for Home, Registration, Shopping Cart, Sign In, and Administration. The main content area is divided into several sections:

- Search:** A search bar with a "Category" dropdown set to "All" and a "Title" input field. A "Search" button is located below the input fields.
- More Search Options:** Includes links for "Advanced search" and "Categories". Under "Categories", there are links for "Programming", "Databases", and "HTML & Web design".
- Weekly Specials:** A blue header with the text "Free Shipping on orders over \$40". Below this, a promotional message states: "For limited time only, until next Sunday, you can enjoy free shipping. Simply order more than \$40 worth of books and shipping's on us."
- Recommended Titles:** A blue header. The first item is "Web Database Development : Step by Step" by Jim Buyens, priced at 39.99. The book cover is yellow and red. The second item is "MySQL & PHP From Scratch" by Wade Maxfield, priced at 23.99. The book cover is black and green.
- What We're Reading:** A blue header. The section is titled "A Sharp Combination" and describes C# programming. It includes a small image of a book cover for "C# Programming".
- New & Notable:** A blue header. The section features "1001 Web Site Construction Tips and Tricks" priced at 39.95. The book cover is white and blue.
- This Week's Featured Books:** A blue header. The section features "Flash 4 Magic" with a book cover showing a computer screen.

... and one flaw: can change any profile (incl. admin password)

Experimental Results

Simplifications used:

- ignored HTML output (irrelevant for control flow)

- ignored some sanitizing functions, assumed all parameters exist

Full bookstore code (22 kLOC)

- 90 seconds for model construction

- 600 lines ASLan++ model, 171 transition rules

For targeted flaw (change admin password):

- 2 components only, 140 lines ASLan++, 22 transition rules

- CL-AtSe model checker *finds flaw in 8 sec.*

- after code correction, model checker shows absence of flaw

Conclusions

Building analyzable models of web applications is possible
by *adapting translation to specific technologies* (e.g., JSP)

Flaws in actual applications can be found/confirmed.

Abstraction is key for obtaining models that can be handled

Next steps:

- extend language support (e.g. exceptions)

- more abstraction (selecting components, predicate abstraction)

- optimize representation (working with sets, strings, etc.)