Identifying Inter-Component Control-Flow in Web Applications

William G.J. Halfond

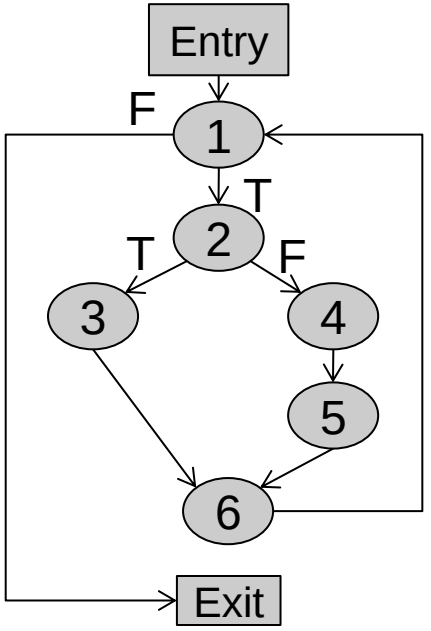
University of Southern California

halfond@usc.edu

Definition: Control-flow

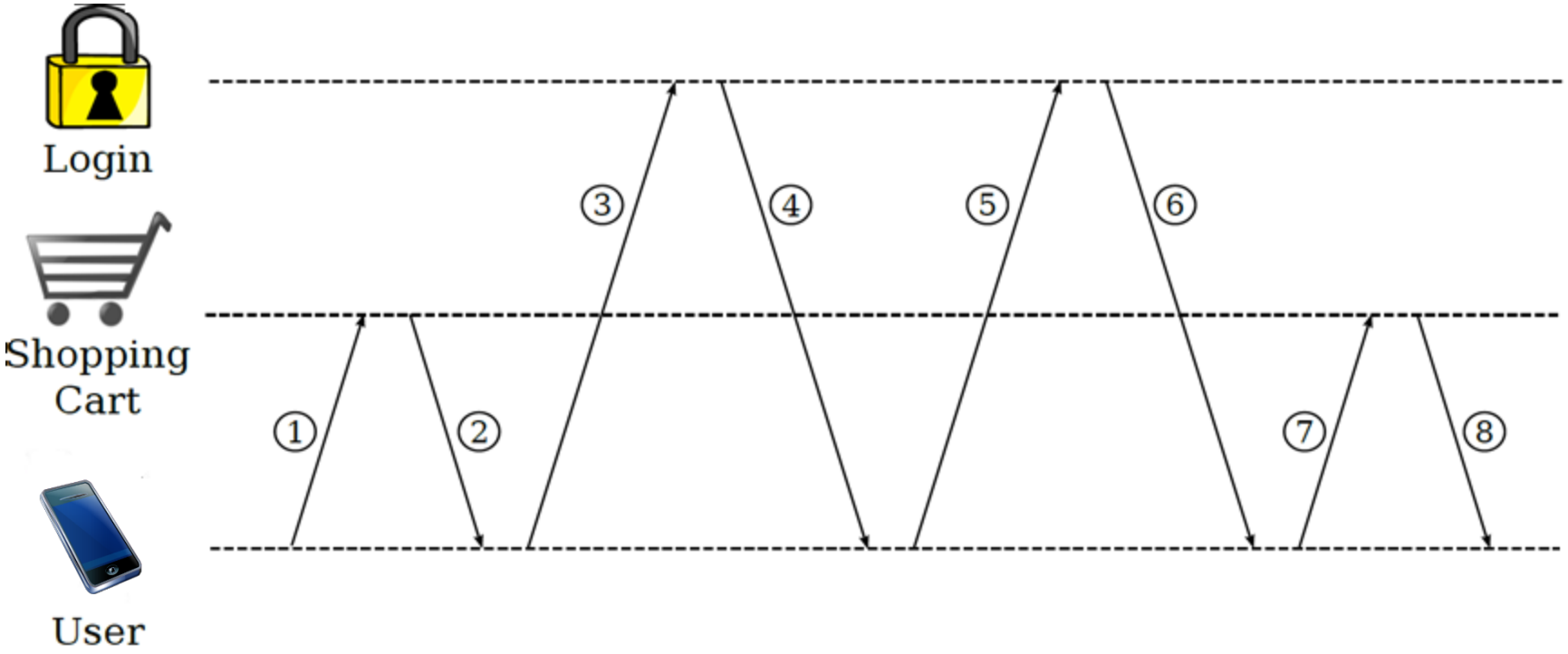
Relationship that shows which statements may execute after each other. I.e sequencing information.

```
function OddorEven(int a)
1.   for(int i=0; i++; i<a){
2.     if (i % 2 == 0) then
3.        print(i + " is even.")
4.     else
5.        print(i + " is odd.")
6.   }
```



Motivating Scenario

Verify user will be logged in before accessing shopping cart



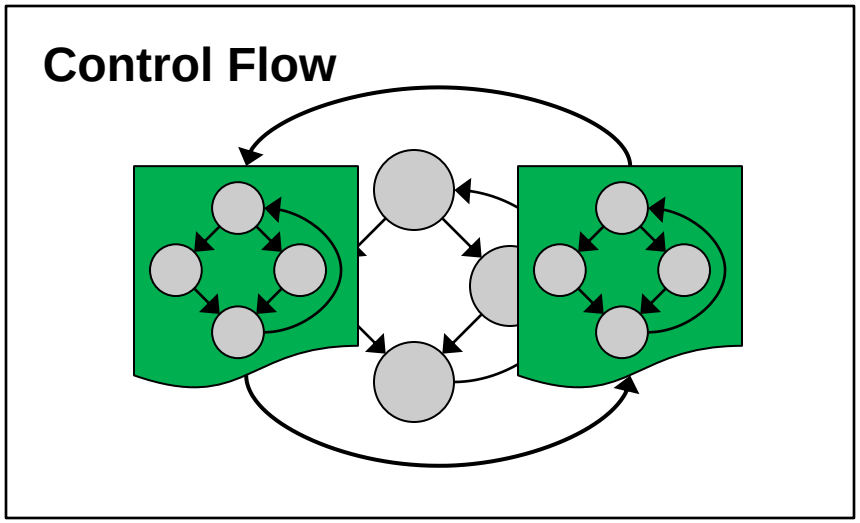
1—8 represent control flow in web applications

Types and Sources of Control-flow

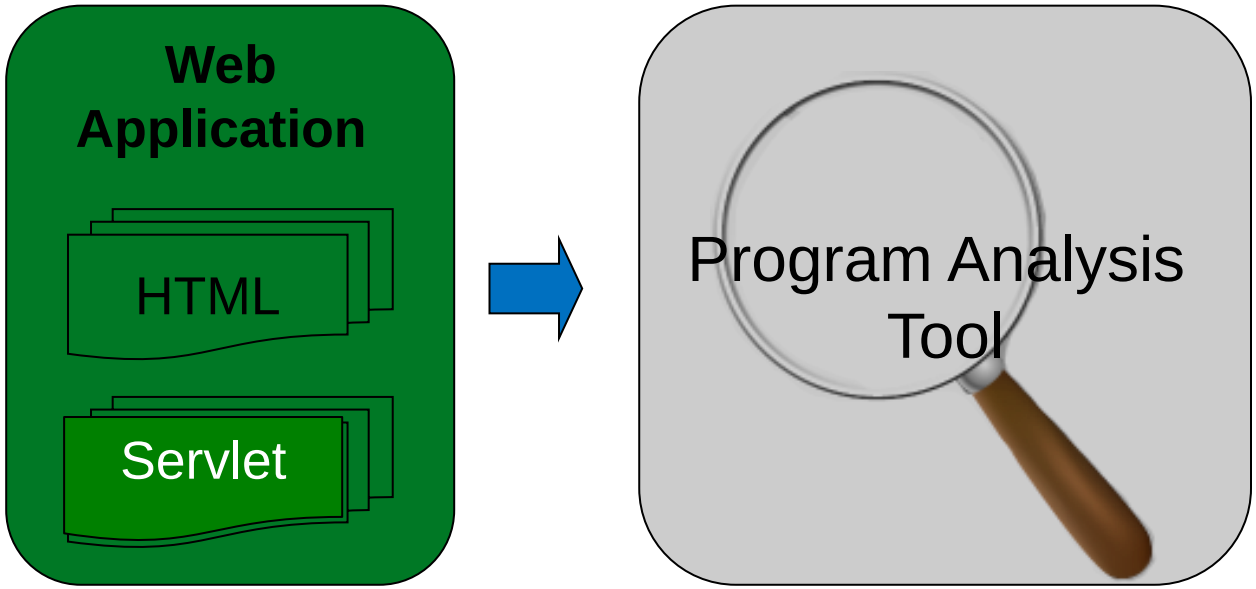
1. Dynamically Generated HTML
2. JavaScript
3. HTTP Commands
4. Component Inclusion
5. Direct Entry/Access

Why Not Use….

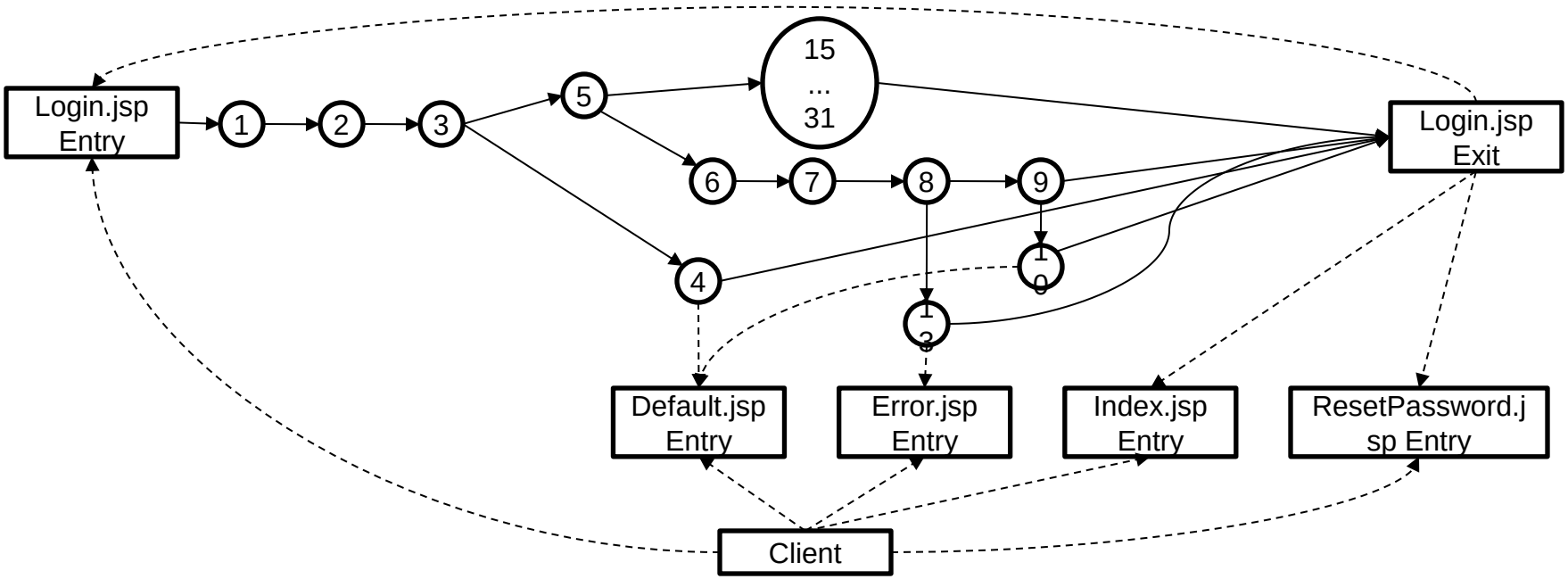
- Manual specification
- Web crawlers
- Traditional control-flow analysis



The Approach: Overview



The Approach: Overview



The Approach: Running Example

```
void service(Request req, Response resp)
1. JspWriter out = resp.getOutputStream();
2. String session = req.getParam("session");
3. if (isValidSession(session))
4.   sendHttpCmd(resp, 302, "Default.jsp");
5. elsif (session.equals("login"))
6.   String login = req.getParam("uname");
7.   String password = req.getParam("pword");
8.   if (isClean(login) && isClean(pword))
9.     if (loginOK(uname, pword))
10.      sendHttpCmd(resp, 302, "Default.jsp");
12.   else
13.      sendHttpCmd(resp, 303, "Error.jsp");
15. else
16.   out.print("<html><body>");
17.   out.print("<script language='JavaScript'>");
18.   out.print("function goBack() {");
19.   out.print("window.location.href="Index.jsp";
20.   out.print("}");
21. out.print("</script>");
22. out.print("<h1>Login Page</h1>");
23. out.print("<form method=POST" + "
    action='Login.jsp'>");
24. out.print("<input type=hidden value=" + "'login'
    name=session>");
25. out.print("User:<input type=text name=uname>");
26. out.print("Password:<input type=" + "password
    name=pword>");
27. out.print("<input type=submit value='Login'>");
28. out.print("<input type=submit value='Back'" + "
    onClick='goBack()'>");
29. out.print("</form>");
30. out.print("<a href='Reset.jsp'>" + " Reset
    password</a>");
31. out.print("</body></html>");
```

The Approach: Step-by-Step

1. Use static analysis to identify dynamically generated HTML and JavaScript content
2. Identify parameter values to invocations of HTTP and Component Inclusion commands
3. Identify open entry methods for Direct Entry

(1) Generated Content: Intuition

Use static analysis to determine potential HTML and JavaScript output of each servlet, and then parse that output to identify relevant constructs and tags.

For each method *m*, in each servlet:

1. Identify HTML content of each output statement
2. Group content along a path into HTML fragments
3. Add HTML fragment to *m*'s summary

Combine summaries up to root method

Parse summaries of each root method

(1) Generated Content: Algorithm

$$\texttt{Gen}[n] = \begin{cases} \{\{\}\} & \text{if } n \text{ is method entry} \\ \{n\} & \text{if } n \text{ generates output} \\ \{n\} & \text{if } n \text{ is a callsite} \\ & \text{and target}(n) \text{ has a summary} \\ \{\} & \text{otherwise} \end{cases}$$

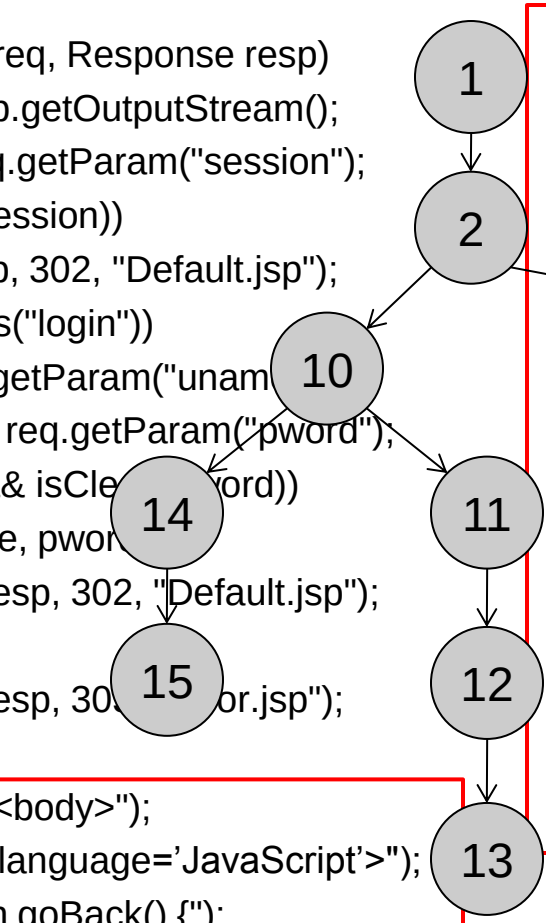
$$\texttt{In}[n] = \bigcup_{p \in pred(n)} \texttt{Out}[p]$$

$$\texttt{Out}[n] = \{p | \forall i \in \texttt{In}[n], p \leftarrow \texttt{append}(i, \texttt{Gen}[n])\}$$

$$\texttt{summary}(m) = \left\{ p | \forall s \in \texttt{Out}[exit(m)] \prod_{n \in s} \texttt{resolve}(n) \right\}$$

(1) Generated Content: Example

```
void service(Request req, Response resp)
1. JspWriter out = resp.getOutputStream();
2. String session = req.getParam("session");
3. if (isValidSession(session))
4.   sendHttpCmd(resp, 302, "Default.jsp");
5. elsif (session.equals("login"))
6.   String login = req.getParam("unam
7.   String password = req.getParam("pword");
8.   if (isClean(login) && isCle       ord))
9.     if (loginOK(uname, pwor
10.      sendHttpCmd(resp, 302, "Default.jsp");
12.   else
13.      sendHttpCmd(resp, 30       or.jsp");
15. else
16.   out.print("<html><body>");
17.   out.print("<script language='JavaScript'>");
18.   out.print("function goBack() {");
19.   out.print("window.location.href="Index.jsp";
20.   out.print("}");
21. out.print("</script>");
22. out.print("<h1>Login Page</h1>");
23. out.print("<form method=POST" + "
    action='Login.jsp'>");
24. out.print("<input type=hidden value=" + "'login'
    name=session>");
25. out.print       <input type=text name=uname>");
26. out.print("Password:<input type=" + "password
    name=pword>");
27. out.print       t type=submit value='Login'>");
28. out.print("<input type=submit value='Back'" + "
    onClick='goBack()'>");
29. out.print       m>");
30. out.print       ref='Reset.jsp'>" + " Reset
    password</a>");
31.       print("</body>       >");
```



Identify HTML producing
ements, i.e GEN sets

(2) Parameter Analysis: Intuition

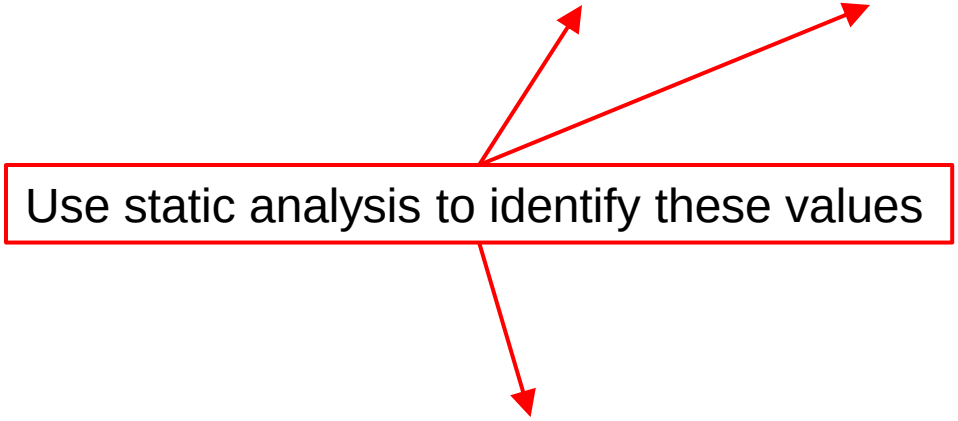
1. Identify invocations in code that call*
    1. HTTP commands
    2. Component Inclusion
2. Analyze parameters to determine value**
3. Interpret semantics of parameter values

*Identify indirect invocations as well and use method summarization.

**String analysis and reaching definitions

(2) Parameter Analysis: Example

resp.sendHttpMessage(302, “Login.jsp”);



Servlet.include(“Common.jsp”)

(2) Parameter Analysis: Example

```
sendHttpCmd(resp, 302, "Default.jsp");
```

Create method summary with placeholders, then substitute in actual values when we find an invocation of the summarized method.

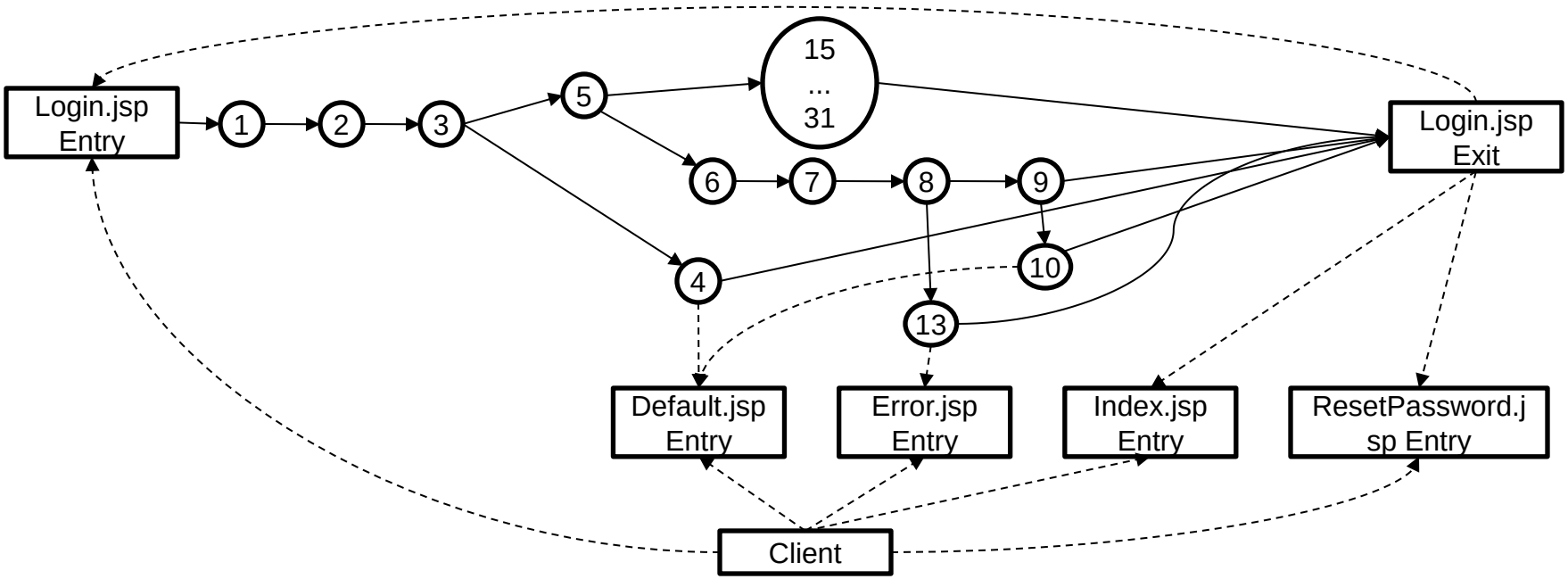
```
void sendHttpCmd(Response resp, int code, String msg)
   String location = "Location: ";
   location += urlEncode(msg);
   location += "\n\n";
   resp.sendHttpMessage(code, location);
```

(3) Entry Points

Identify and mark methods that represent methods that can be invoked and provided with input by the user.

Examples: doPost(), doGet(), _service()

Inter-Component Control-flow Graph



The Approach: Running Example

```
void service(Request req, Response resp)
1. JspWriter out = resp.getOutputStream();
2. String session = req.getParam("session");
3. if (isValidSession(session))
4.   sendHttpCmd(resp, 302, "Default.jsp");
5. elsif (session.equals("login"))
6.   String login = req.getParam("uname");
7.   String password = req.getParam("pword");
8.   if (isClean(login) && isClean(pword))
9.     if (loginOK(uname, pword))
10.      sendHttpCmd(resp, 302, "Default.jsp");
12.   else
13.      sendHttpCmd(resp, 303, "Error.jsp");
15. else
16.   out.print("<html><body>");
17.   out.print("<script language='JavaScript'>");
18.   out.print("function goBack() {");
19.   out.print("window.location.href="Index.jsp";
20.   out.print("}");
21. out.print("</script>");
22. out.print("<h1>Login Page</h1>");
23. out.print("<form method=POST" + "
    action='Login.jsp'>");
24. out.print("<input type=hidden value=" + "'login'
    name=session>");
25. out.print("User:<input type=text name=uname>");
26. out.print("Password:<input type=" + "password
    name=pword>");
27. out.print("<input type=submit value='Login'>");
28. out.print("<input type=submit value='Back'" + "
    onClick='goBack()'>");
29. out.print("</form>");
30. out.print("<a href='Reset.jsp'>" + " Reset
    password</a>");
31. out.print("</body></html>");
```

Evaluation: Research Questions

RQ1: Time to analyze web applications.

RQ2: Precision of control-flow information.

RQ3: Recall of control-flow information.

Implemented approach in **ICE** (Inter-component Control-flow Extractor)

- Soot, Indus, JSA, HTMLParser, Rhino for underlying analyses

Subject Applications

Application	LOC	Classes	Servlets
Bookstore	19,402	28	27
Classifieds	10,702	18	18
Daffodil	18,706	119	70
Employee Dir.	5,529	11	9
Events	7,164	13	12
Filelister	8,671	41	10
Portal	16,089	28	27
Webmail	17,078	81	24

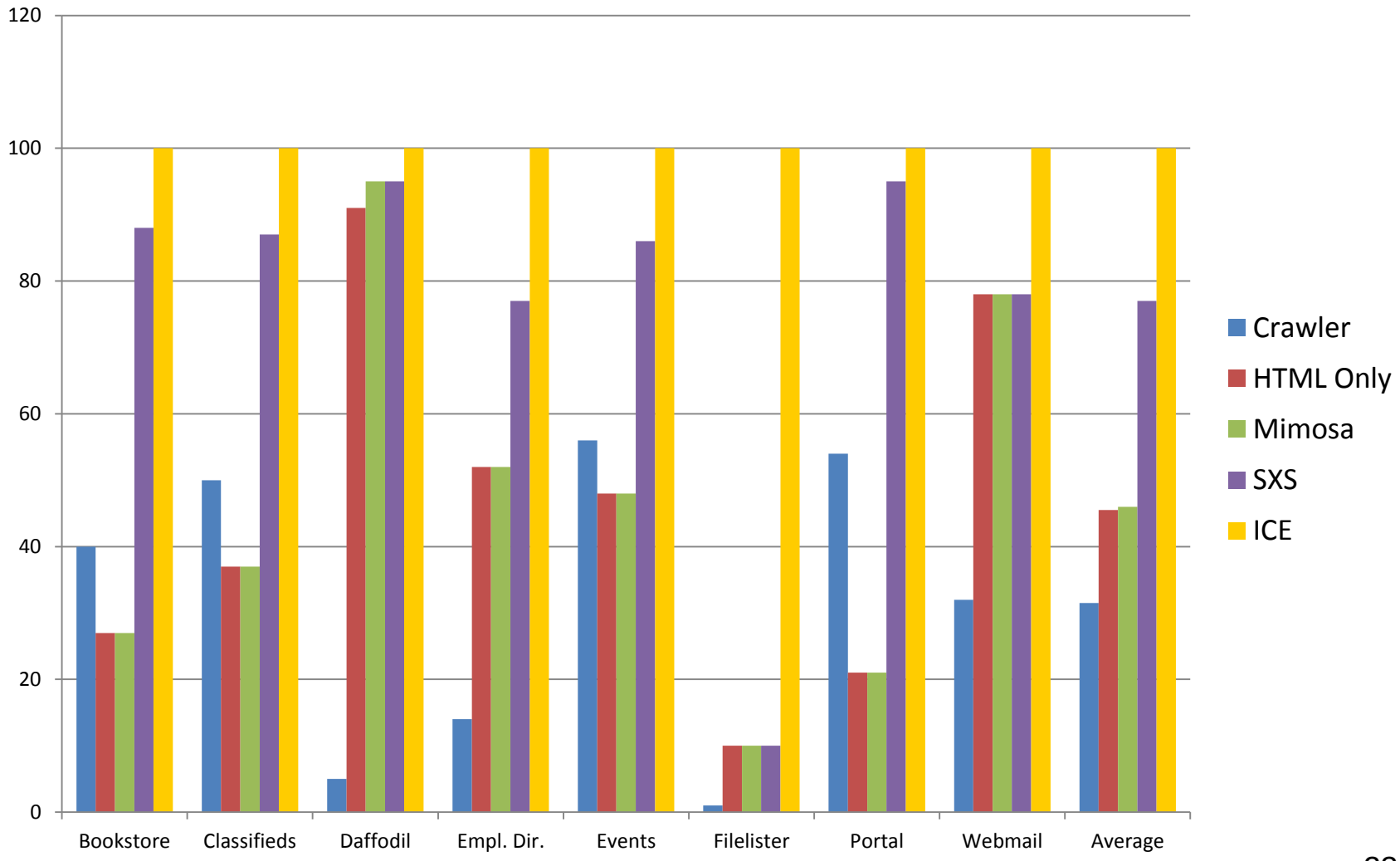
Other Approaches

- **Crawler:** Automated exploration of web pages at runtime, based on Crawljax and a traditional static web page crawler
- **HTML only:** Only uses step 1 of the approach, identifying dynamic HML content
- **Mimosa:** Static analysis tool for detecting workflow attacks
- **SXS:** Static analysis tool for detecting access control vulnerabilities

Evaluation Methodology

1. Manually built ground truth for each of the subject applications
2. Run each approach on each subject – measure runtime
3. Compare identified control-flow with the ground truth for each subject
    1. Calculate precision
    2. Calculate recall

Results for Recall



Summary

- Control-flow is key to verifying complex behaviors in web applications
- Developed static analysis to identify different types of control-flow
- Empirical evaluation shows
  - High recall and precision
  - Out-performs existing static analyses

Thank you