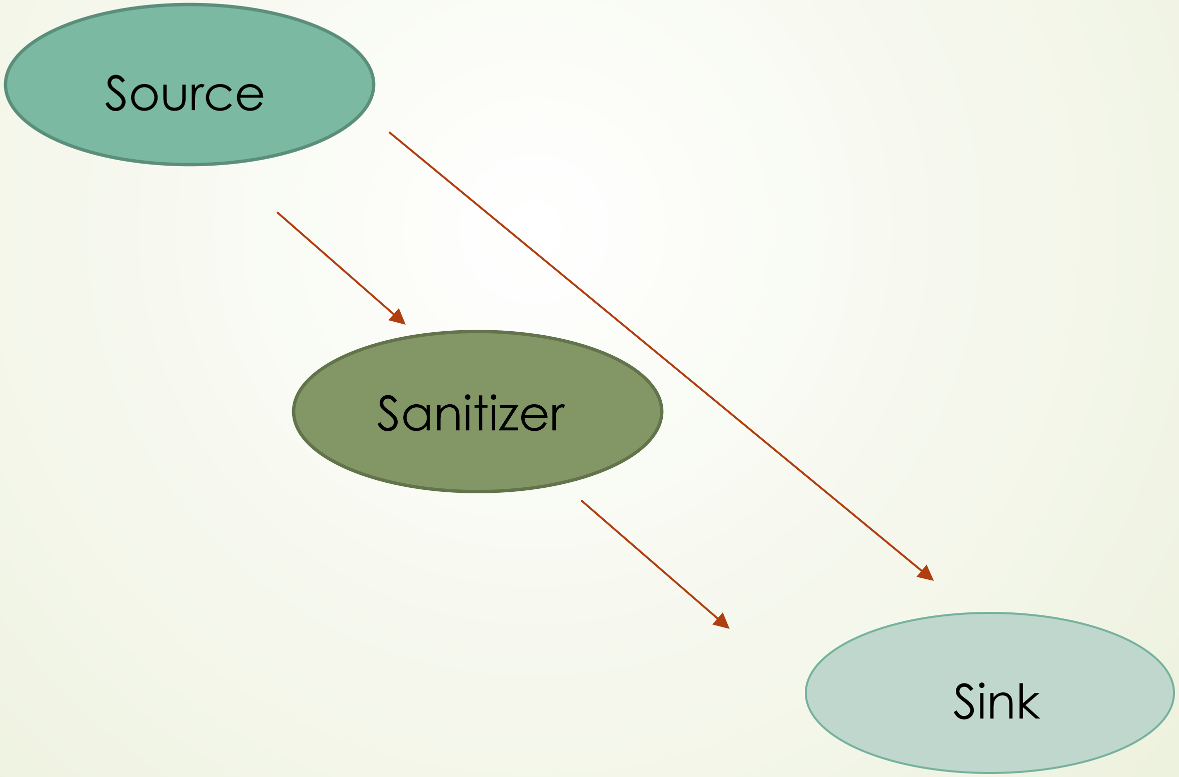
TAJ: Effective Taint Analysis of Web Applications

Omer Tripp, Marco Pistoia, Stephen Fink, Manu Sridharan, Omri Weisman

Published in PLDI 2009

Presented by Dong Chen

Recall: Taint Analysis



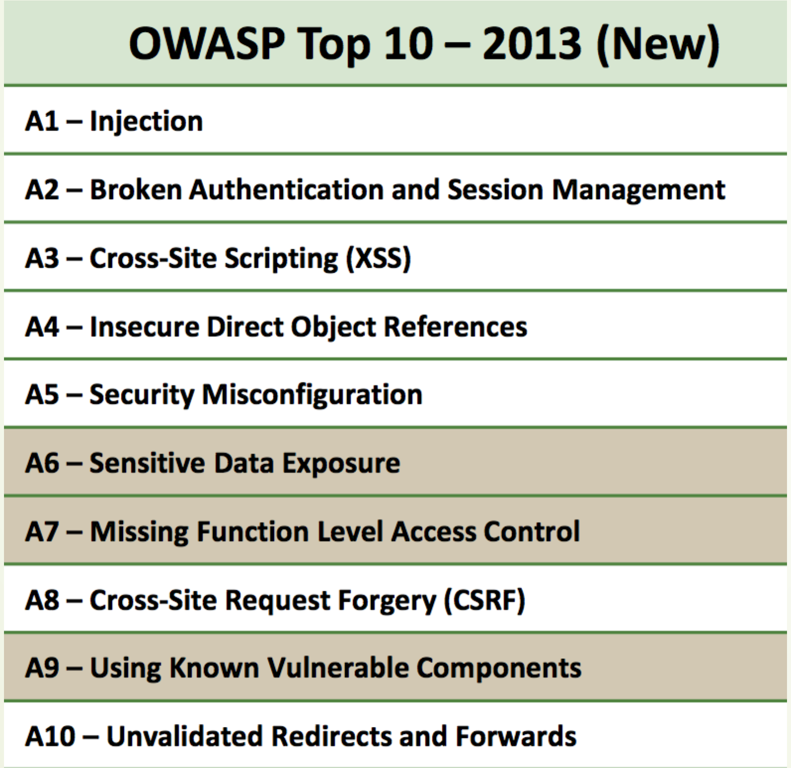
Outline

- Background
- Motivation
- Approach
- Evaluation
- Conclusion

Outline

- **Background**
- Motivation
- Approach
- Evaluation
- Conclusion

OWASP Top Ten Security Vulnerabilities



OWASP Top 10 – 2013 (New)
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

http://www.owasp.org

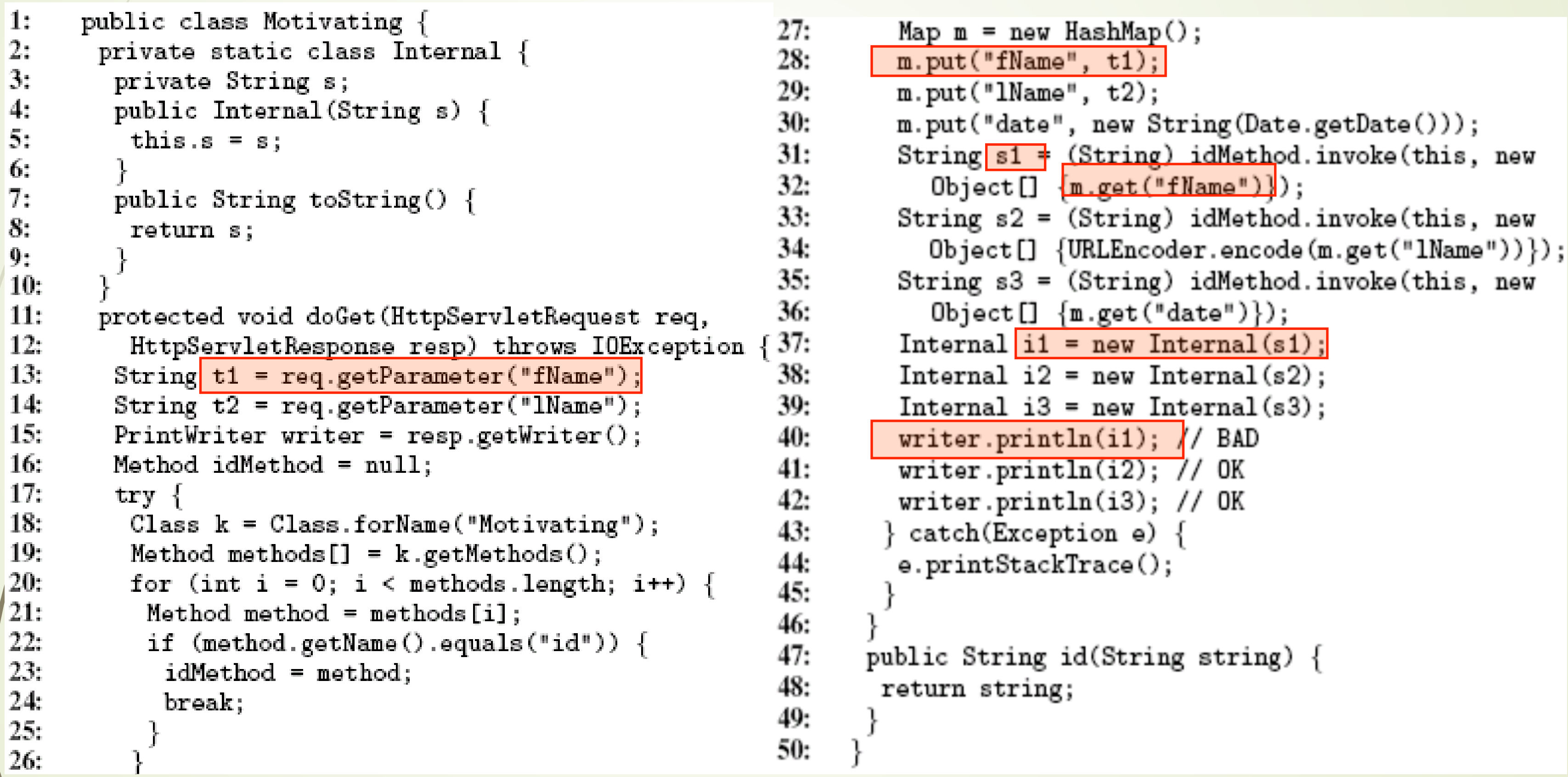
Existing solutions

- Type systems:
  - Complex, conservative, require code annotations
- Slicing:
  - Has not been shown to scale to large applications

Outline

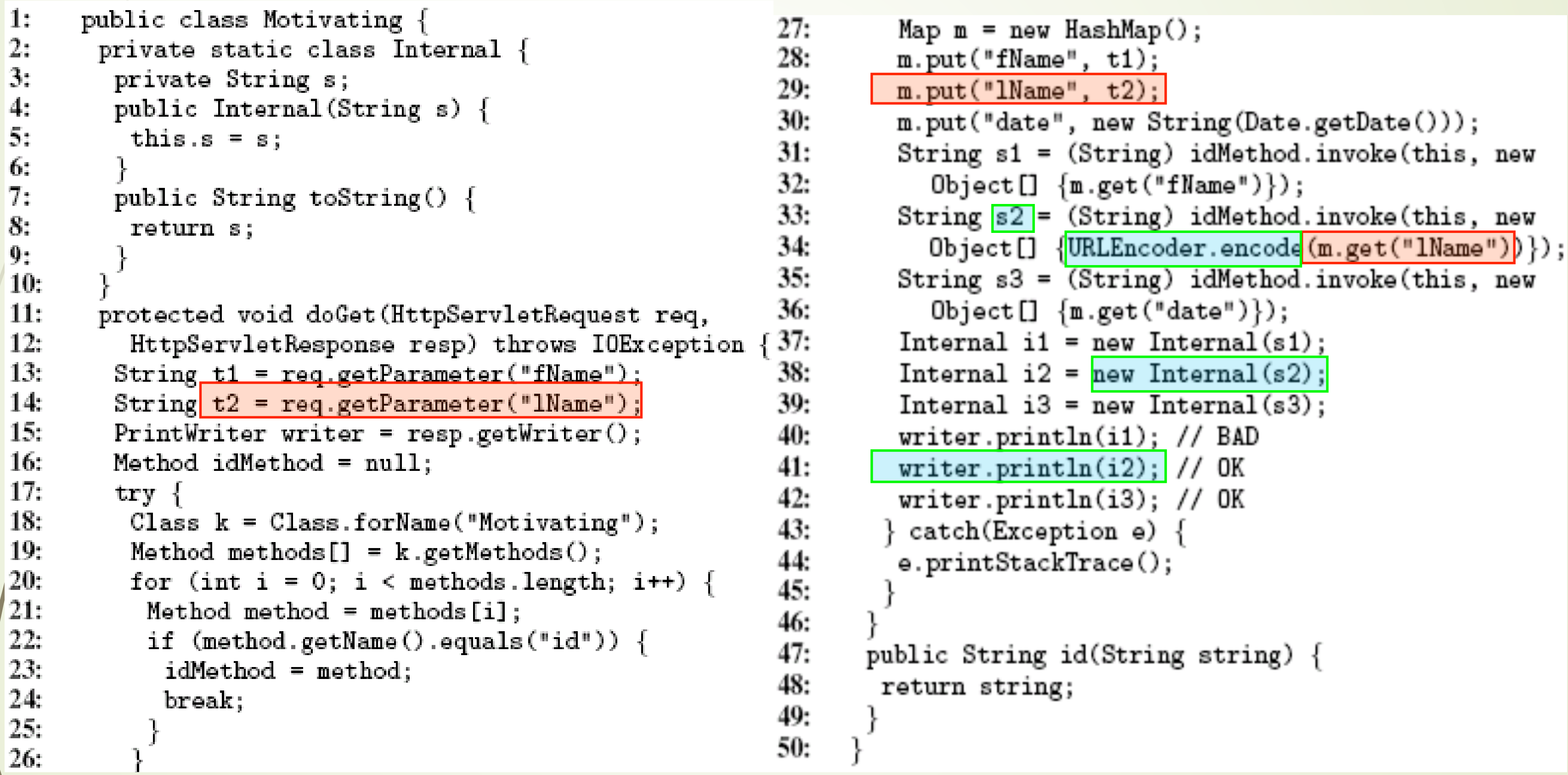
- Background
- **Motivation**
- Approach
- Evaluation
- Conclusion

Motivating Example



```java
public class Motivating {
 private static class Internal {
  private String s;
  public Internal(String s) {
   this.s = s;
  }
  public String toString() {
   return s;
  }
 }
 protected void doGet(HttpServletRequest req,
   HttpServletResponse resp) throws IOException {
  String t1 = req.getParameter("fName");
  String t2 = req.getParameter("lName");
  PrintWriter writer = resp.getWriter();
  Method idMethod = null;
  try {
   Class k = Class.forName("Motivating");
   Method methods[] = k.getMethods();
   for (int i = 0; i < methods.length; i++) {
    Method method = methods[i];
    if (method.getName().equals("id")) {
     idMethod = method;
     break;
    }
   }
   Map m = new HashMap();
   m.put("fName", t1);
   m.put("lName", t2);
   m.put("date", new String(Date.getDate()));
   String s1 = (String) idMethod.invoke(this, new
     Object[] {m.get("fName")});
   String s2 = (String) idMethod.invoke(this, new
     Object[] {URLEncoder.encode(m.get("lName"))});
   String s3 = (String) idMethod.invoke(this, new
     Object[] {m.get("date")});
   Internal i1 = new Internal(s1);
   Internal i2 = new Internal(s2);
   Internal i3 = new Internal(s3);
   writer.println(i1); // BAD
   writer.println(i2); // OK
   writer.println(i3); // OK
  } catch(Exception e) {
   e.printStackTrace();
  }
 }
 public String id(String string) {
  return string;
 }
}
```

Motivating Example



```
public class Motivating {
 private static class Internal {
  private String s;
  public Internal(String s) {
   this.s = s;
  }
  public String toString() {
   return s;
  }
 }
 protected void doGet(HttpServletRequest req,
   HttpServletResponse resp) throws IOException {
  String t1 = req.getParameter("fName");
  String t2 = req.getParameter("lName");
  PrintWriter writer = resp.getWriter();
  Method idMethod = null;
  try {
   Class k = Class.forName("Motivating");
   Method methods[] = k.getMethods();
   for (int i = 0; i < methods.length; i++) {
    Method method = methods[i];
    if (method.getName().equals("id")) {
     idMethod = method;
     break;
    }
   }
   Map m = new HashMap();
   m.put("fName", t1);
   m.put("lName", t2);
   m.put("date", new String(Date.getDate()));
   String s1 = (String) idMethod.invoke(this, new
     Object[] {m.get("fName")});
   String s2 = (String) idMethod.invoke(this, new
     Object[] {URLEncoder.encode(m.get("lName"))});
   String s3 = (String) idMethod.invoke(this, new
     Object[] {m.get("date")});
   Internal i1 = new Internal(s1);
   Internal i2 = new Internal(s2);
   Internal i3 = new Internal(s3);
   writer.println(i1); // BAD
   writer.println(i2); // OK
   writer.println(i3); // OK
  } catch(Exception e) {
   e.printStackTrace();
  }
 }
 public String id(String string) {
  return string;
 }
}
```

Outline

- Background
- Motivation
- **Approach**
- Evaluation
- Conclusion

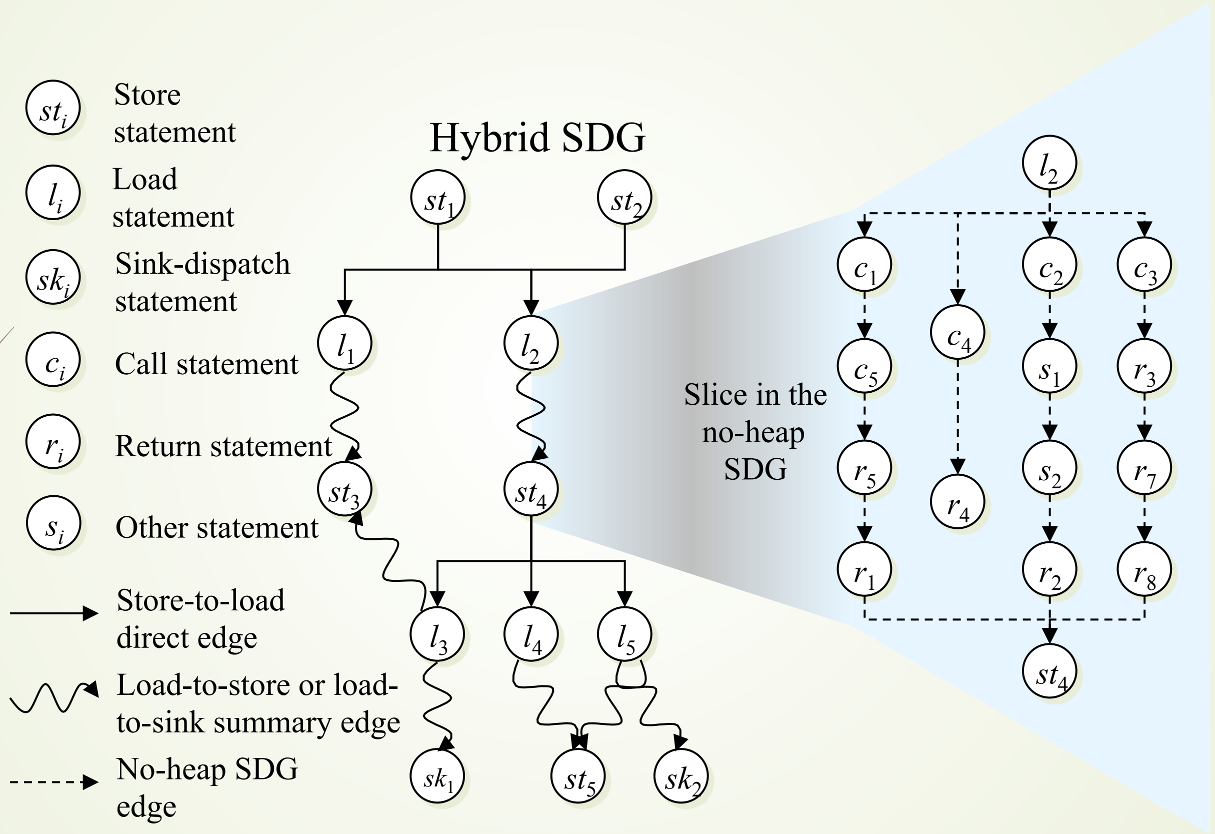
TAJ

- Consists of 2 stages:
    - Pointer analysis
    - Slicing algorithm
- Effective reports
- Efficient behavior under restricted budget

Pointer analysis and call-graph construction

- Pointer analysis is a variant of Andersen's analysis
- Custom context-sensitivity policy:
  - Unlimited-depth object sensitivity for Java collections
  - One level of call-string context for factory methods
  - One level of call-string context for taint APIs
- Pointer analysis of TAJ is field sensitive

Hybrid thin slicing



Direct edges: computed based on preliminary pointer analysis
Summary edges: computed using no-heap SDG

Eliminating Redundant Reports

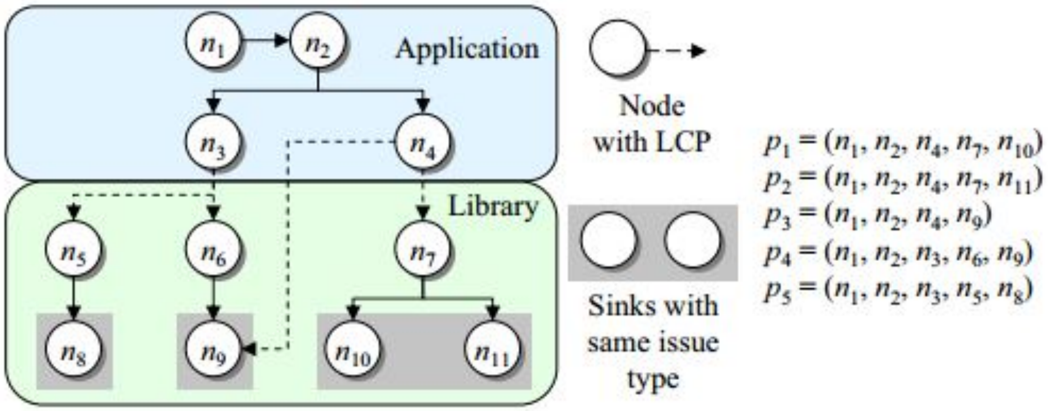

n_1
n_2
Application
n_3
n_4
Library
n_5
n_6
n_7
n_8
n_9
n_{10}
n_{11}
Node with LCP
Sinks with same issue type
$p_1 = (n_1, n_2, n_4, n_7, n_{10})$
$p_2 = (n_1, n_2, n_4, n_7, n_{11})$
$p_3 = (n_1, n_2, n_4, n_9)$
$p_4 = (n_1, n_2, n_3, n_6, n_9)$
$p_5 = (n_1, n_2, n_3, n_5, n_8)$

Figure 3. Call Graph Illustrating the LCP Concept

Example:

1. Use p1 and p2
2. Use p3 and p4

Priority-driven Call-graph Construction

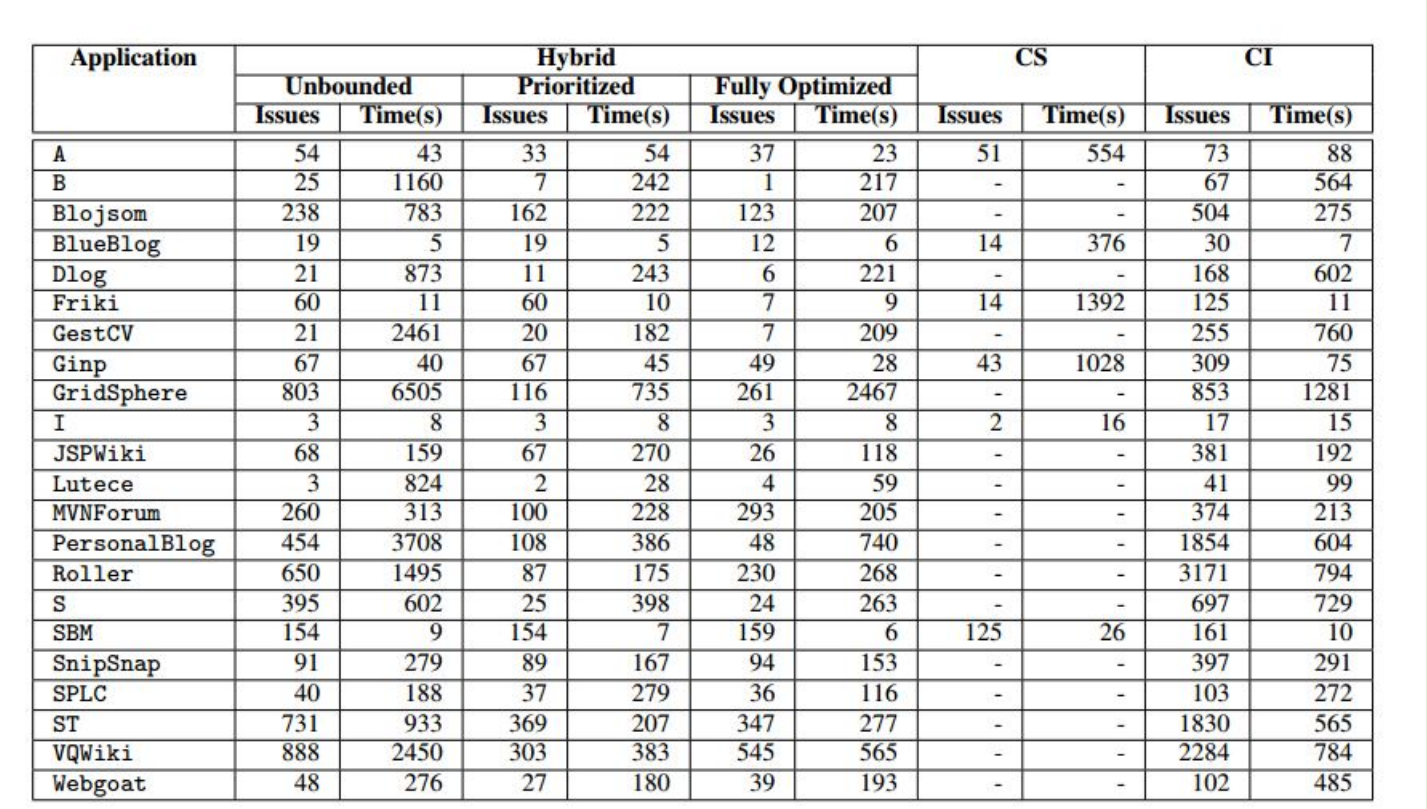
- Priority queue used to govern call-graph growth
- Sources are assigned priority 0, others maxNodes
- Recursively, for each "neighbor" *t* of node *n*: $pr(t) = \min\{(pr(n) + 1), pr(t)\}$
- Propagation process runs to a fixed point
- "Locality-of-taint" principle

Outline

- Background
- Motivation
- Approach
- **Evaluation**
- Conclusion

Evaluation

- Performance

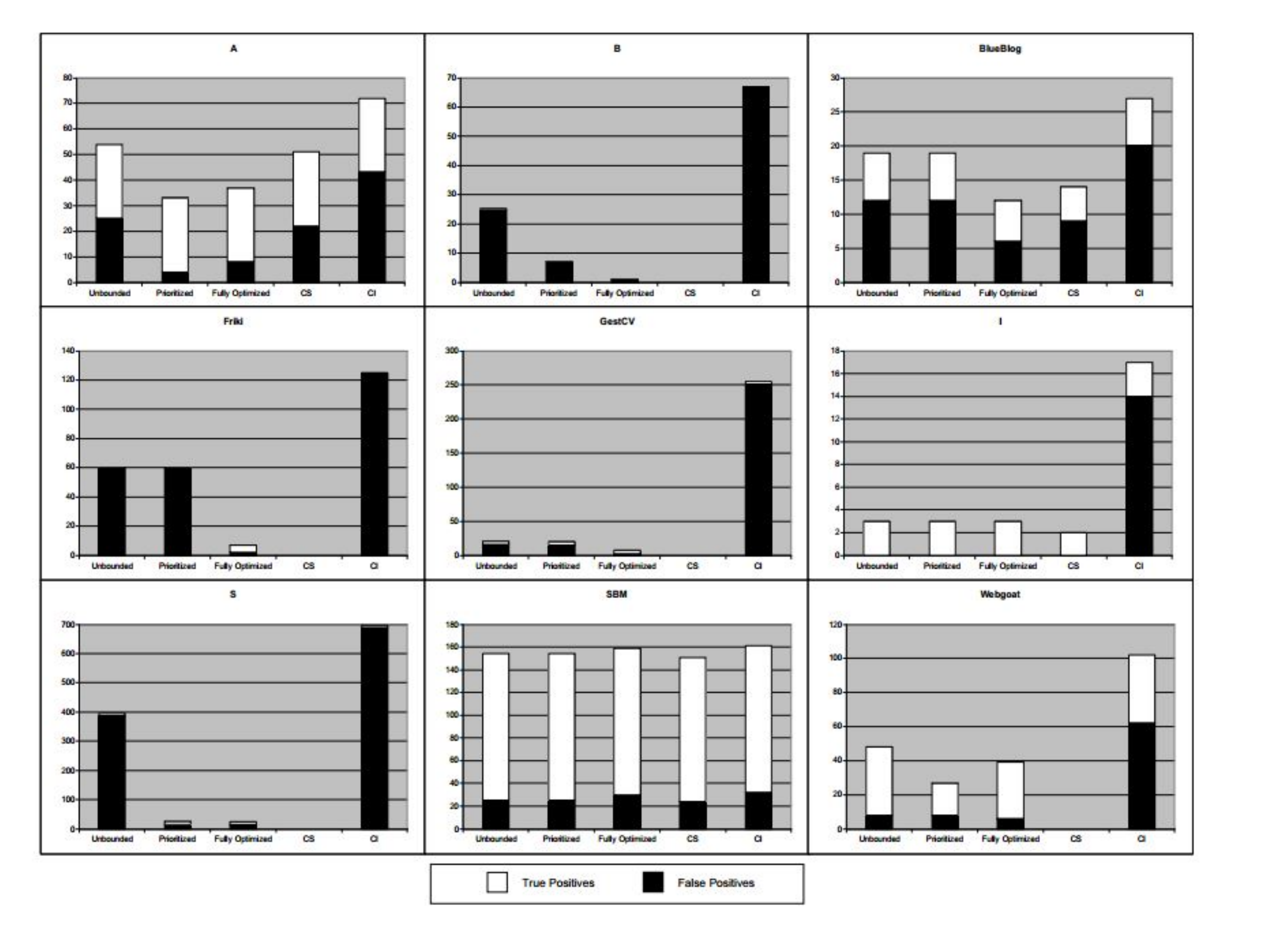


Application	Hybrid						CS		CI	
	Unbounded		Prioritized		Fully Optimized					
	Issues	Time(s)	Issues	Time(s)	Issues	Time(s)	Issues	Time(s)	Issues	Time(s)
A	54	43	33	54	37	23	51	554	73	88
B	25	1160	7	242	1	217	-	-	67	564
Blojsom	238	783	162	222	123	207	-	-	504	275
BlueBlog	19	5	19	5	12	6	14	376	30	7
Dlog	21	873	11	243	6	221	-	-	168	602
Friki	60	11	60	10	7	9	14	1392	125	11
GestCV	21	2461	20	182	7	209	-	-	255	760
Ginp	67	40	67	45	49	28	43	1028	309	75
GridSphere	803	6505	116	735	261	2467	-	-	853	1281
I	3	8	3	8	3	8	2	16	17	15
JSPWiki	68	159	67	270	26	118	-	-	381	192
Lutece	3	824	2	28	4	59	-	-	41	99
MVNForum	260	313	100	228	293	205	-	-	374	213
PersonalBlog	454	3708	108	386	48	740	-	-	1854	604
Roller	650	1495	87	175	230	268	-	-	3171	794
S	395	602	25	398	24	263	-	-	697	729
SBM	154	9	154	7	159	6	125	26	161	10
SnipSnap	91	279	89	167	94	153	-	-	397	291
SPLC	40	188	37	279	36	116	-	-	103	272
ST	731	933	369	207	347	277	-	-	1830	565
VQWiki	888	2450	303	383	545	565	-	-	2284	784
Webgoat	48	276	27	180	39	193	-	-	102	485

Table 3. Experimental Results Comparing between Hybrid Variants and Other Algorithms

Evaluation

- Accuracy



Outline

- Background
- Motivation
- Approach
- Evaluation
- **Conclusion**

Conclusion

- Effective solution for taint analysis of Web applications based on pointer analysis and hybrid thin slicing
- Efficient strategies for analysis under limited budget

Questions