Andromeda:
Accurate and Scalable Security Analysis of Web Applications

Omer Tripp
Tel Aviv University & IBM
omert@il.ibm.com

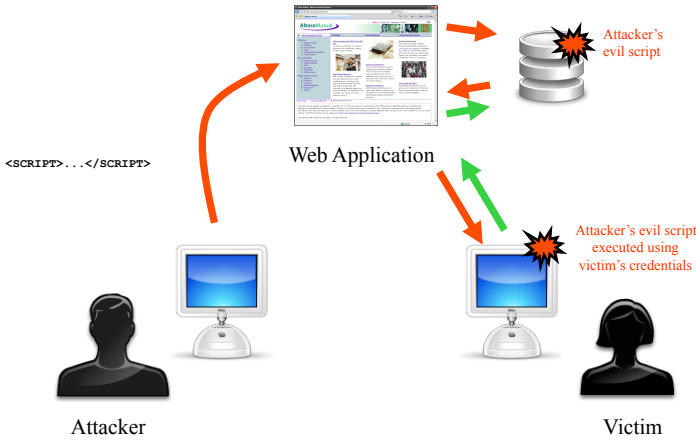
Marco Pistoia
IBM T. J. Watson Research Center
pistoia@us.ibm.com

Patrick Cousot
New York University
pcousot@cs.nyu.edu

Radhia Cousot
École Normale Supérieure
radhia.cousot@ens.fr

Salvatore Guarnieri
University of Washington & IBM
sguarni@us.ibm.com

XSS



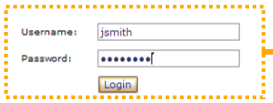
OWASP* Top Ten Security Vulnerabilities

1. Cross-site scripting (XSS)
2. Injection flaws
3. Malicious file executions
4. Insecure direct object reference
5. Cross site request forgery (CSRF)
6. Information leakage and improper error handling
7. Broken authentication and improper session management
8. Unsecure cryptographic storage
9. Unsecure communications
10. Failure to restrict URL accesses

* Open Web Application Security Project (OWASP): http://www.owasp.org

SQL Injection

```
String query = "SELECT * FROM users WHERE name='" +
  userName + "' AND pwd='" + pwd + "'";
```




```
SELECT * FROM users WHERE name='jsmith' AND pwd='Demo1234'
```


```
SELECT * FROM users WHERE name='foo';drop table custid;--' AND pwd=''
```

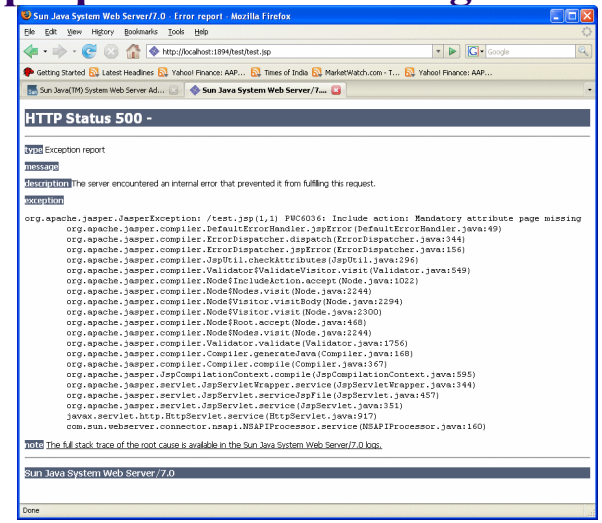
Malicious File Executions

- Web application manage files in the file system
- The name or contents of such files are often obtained from user input
- Maliciously crafted user inputs could cause the execution or deletion of security-sensitive files

Existing Static-Analysis Solutions

- Type systems:
  - Complex, conservative, require code annotations
- Classic slicing:
  - Has not been shown to scale to large applications while maintaining sufficient accuracy

Information Leakage and Improper Error Handling

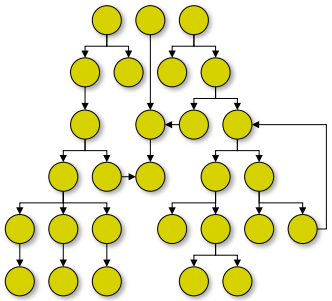


Motivation

- Web applications are large and complex
- Sound analyses
  - If too precise, do not scale well
  - If too imprecise, have too many false positives
- Unsound analyses
  - Have false negatives
  - Are often unstable (extra-sensitivity to program changes)

Intuition behind Andromeda

- Taint analysis can be treated as a demand-driven problem
- This enables lazy computation of vulnerable information flows, instead of eagerly computing a complete data-flow solution



Motivating Example

```
public class Aliasing5 extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
         throws ServletException, IOException {
      StringBuffer buf = new StringBuffer("abc");
      foo(buf, buf, resp, req);
   }

   void foo(StringBuffer buf, StringBuffer buf2, ServletResponse resp,
         ServletRequest req) throws IOException {
      String name = req.getParameter("name");
      buf.append(name);
      PrintWriter writer = resp.getWriter();
      writer.println(buf2.toString()); /* BAD */
   }
}
```

Publications on Andromeda

- FASE 2013 – Andromeda algorithm
  - Omer Tripp, Marco Pistoia, Patrick Cousot, Radhia Cousot, Salvatore Guarnieri, "Andromeda: Accurate and Scalable Security Analysis of Web Applications"
- OOPSLA 2011 – Integration with Framework for Frameworks (F4F)
  - Manu Sridharan, Shay Artzi, Marco Pistoia, Salvatore Guarnieri, Omer Tripp, Ryan Berg, "F4F: Taint Analysis of Framework-based Web Applications"
- ISSTA 2011 (1) – Andromeda for JavaScript
  - Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, Ryan Berg, "Saving the World Wide Web from Vulnerable JavaScript"
- ISSTA 2011 (2) – Andromeda as the basis for String Analysis (ACM SIGSOFT Distinguished Paper Award)
  - Takaaki Tateishi, Marco Pistoia, Omer Tripp, "Path- and Index-sensitive String Analysis based on Monadic Second-order Logic"
- IBM Journal on Research and Development 2013 – Permission analysis for Android applications
  - Dragoş Sbîrlea, Michael G. Burke, Salvatore Guarnieri, Marco Pistoia, Vivek Sarkar, "Automatic Detection of Inter-application Permission Leaks in Android Applications"

Contributions of Andromeda

- Scalable and sound demand-driven taint analysis
- Modular analysis
- Incremental analysis
- Framework and library support
- Multiple language support (Java, .NET, JavaScript)
- Inclusion in an IBM product: IBM Security AppScan Source

High-level Algorithm

- Input: Web application plus supporting rules
  - {(*Sources*, *Sinks*, *Sanitizers*)}
- Build class hierarchy
- Construct CHA-based call graph with intra-procedural type-inference optimization
- Perform data-flow analysis (explained next)
- Report any flow from a source to a sink not intercepted by a sanitizer in the same rule

Modularity of the Analysis

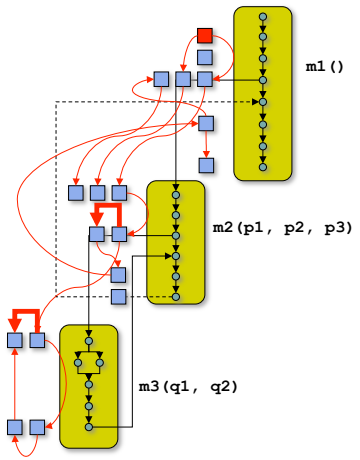
- Runs on data flow (def-to-use)
- Produces and uses pre-compiled models
  - Format:
    `<method, entry>` → `<method, exit>`
  - Example:
    `<m, v2.f.g>` → `<m, v1.h>`

Abstract Domain

- Consists of triplets:
  - Method where Static Single Assignment (SSA) variable is defined
  - SSA variable ID
  - Access path
- Inputs form a lattice according to subsumption relation defined on access paths, *e.g.*:
  `o.* ≥ o.f.* ≥ o.f.g`
  - The `*` symbol represents any feasible sub-path
  - Array load/store semantics is applied to arrays, maps, session objects, etc.

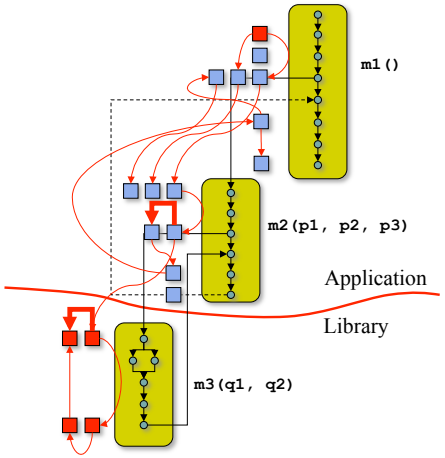
A Novel Approach to Taint Analysis

- Start from taint sources
- Propagate taint intra-procedurally through def-to-use
- Inter-procedurally propagate taint forward and record constraints in callees
- Record constraints on call sites, recursively (allows for polymorphism)
- Resolve aliasing by going back to allocation sites
- In the final *constraint-propagation graph*, detect paths between sources and sinks not intercepted by sanitizers



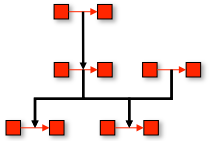
Modular Analysis

- Persist constraint edges at library entrypoints
- Constraint edges are mapped to contexts
- During analysis time, the constraint edges specific to a particular context are used
- Summaries are source-, sink- and sanitizer-specific



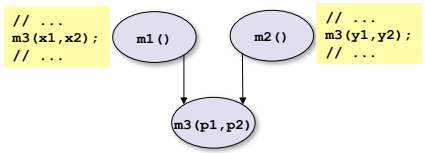
Incremental Analysis

- A *taint constraint* is an edge in the constraint-propagation graph
- The *support graph* records how constraints were learned (*i.e.*, based on which other constraints)
- Facts learned in a scope that underwent change are transitively invalidated
- Preconditions recomputed
- Fixed-point analysis recommenced



Backward Propagation

- Pushes constraints back to callers
  - Infinite context sensitivity
  - Polymorphism with respect to taint
- The constraint `p1.f.g` → `p2.h` in `m3` is propagated to `m1` and `m2` (and, recursively, to their callers)
  - `x1.f.g` → `x2.h`
  - `y1.f.g` → `y2.h`



Integration with F4F

- F4F (OOPSLA 2011) analyzes code and metadata of frameworks and represents them in artifacts written in an XML-like language
- Andromeda translates those artifacts into legal Java code that – from a data-flow perspective – is equivalent to the original framework code
- New code is human-readable and reusable by other analyzers
- New code is compiled and added to the analysis scope

Experimental Results*

	ANDROMEDA	TAJ
Average TPs	82%	68%
Average FPs	12%	30%
Average Unknowns	6%	2%

Change Type	**Response Time (s)**			
	AltoroJ		Webgoat	
	Deletion	**Addition**	**Deletion**	**Addition**
Taint-propagator statement	2	2.2	1.9	2.2
Security sink	0.5	2	1.9	2.5
Security source	2.1	2.1	1.8	3.2
Irrelevant statement	1.9	2	2.5	2.8
Relevant method	2.2	1.9	1.8	2.7
Irrelevant method	2.2	1.7	1.7	1.7

* More details in paper

Thank You!

pistoia@us.ibm.com

Conclusion

- The notorious scalability barrier finally lifted without compromising soundness
- Incremental analysis is a great promise for developers
- Production summaries already generated