



Securing Web Applications with Information Flow Tracking

Monica Lam
Stanford University

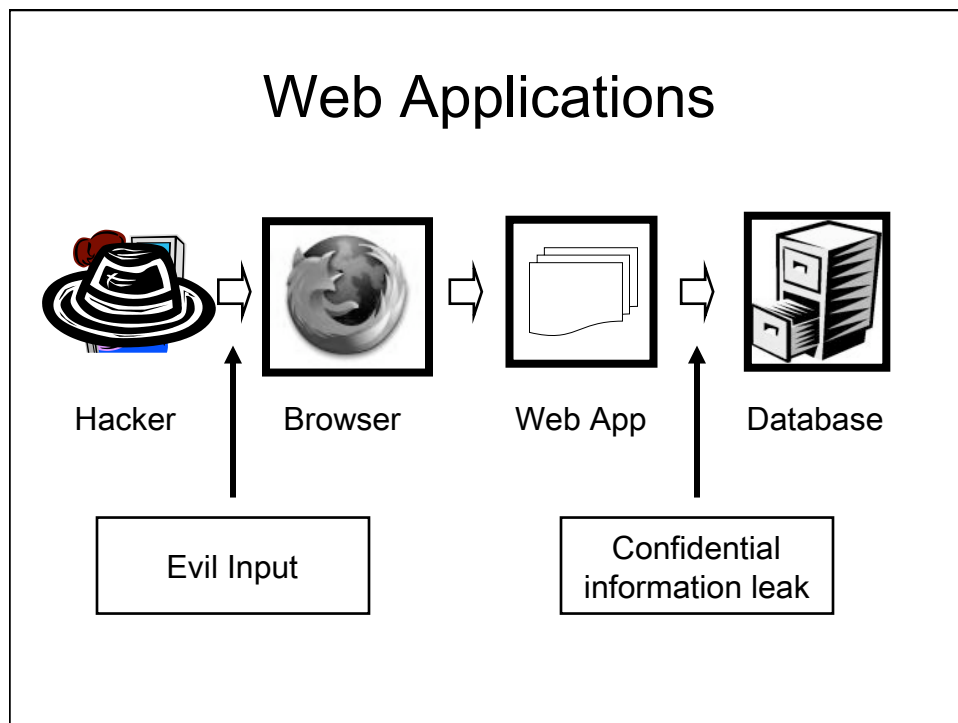
with Michael Martin, Benjamin Livshits, John Whaley,
Michael Carbin, Dzin Avots, Chris Unkel

Web Application Vulnerabilities

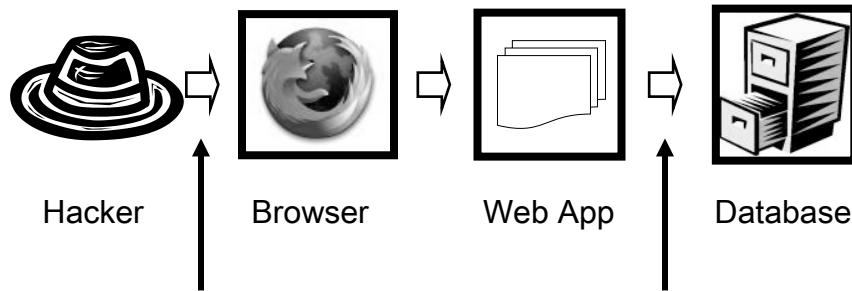
- 50% databases had a security breach
[Computer crime & security survey, 2002]
- 92% Web applications are vulnerable
[Application Defense Center, 2004]
- 48% of all vulnerabilities Q3-Q4, 2004
[Symantec May, 2005]

Top Ten Security Flaws in Web Applications [OWASP]

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



SQL Injection Errors



Give me Bob's credit card #
Delete all records

Happy-go-lucky SQL Query

User supplies: *name*, *password*

Java program:

String query =

```
"SELECT UserID, Creditcard FROM CCRec  
WHERE Name = "
```

```
+ name + " AND PW = "
```

```
+ password
```

Fun with SQL

“ — ”: “the rest are comments” in Oracle SQL

```
SELECT UserID, CreditCard FROM CCRec
```

WHERE:

```
Name = bob AND PW = foo
```

```
Name = bob— AND PW = x
```

```
Name = bob or 1=1— AND PW = x
```

```
Name = bob; DROP CCRec— AND PW = x
```

Vulnerabilities in Web Applications

Inject

Parameters

Hidden fields

Headers

Cookie poisoning

X

Exploit

SQL injection

Cross-site scripting

HTTP splitting

Path traversal

Key: Information Flow

A Simple SQL Injection Pattern

```
o = req.getParameter ( );  
stmt.executeQuery ( o );
```

In Practice

```
ParameterParser.java:586
String session.ParameterParser.getRawParameter(String name)

public String getRawParameter(String name)
    throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + " not found");
    } else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + " was empty");
    }
    return values[0];
}
```

```
ParameterParser.java:570
String session.ParameterParser.getRawParameter(String name, String def)

public String getRawParameter(String name, String def) {
    try {
        return getRawParameter(name);
    } catch (Exception e) {
        return def;
    }
}
```

In Practice (II)

```
ChallengeScreen.java:194
Element lessons.ChallengeScreen.doStage2(WebSession s)

String user = s.getParser().getRawParameter( USER,
"" );
StringBuffer tmp = new StringBuffer();
tmp.append("SELECT cc_type, cc_number from user_data
WHERE userid = '");
tmp.append(user);
tmp.append("'");
query = tmp.toString();
Vector v = new Vector();
try
{
    ResultSet results = statement3.executeQuery( query );
    ...
}
```

PQL: Program Query Language

```
o = req.getParameter ( );  
stmt.executeQuery ( o );
```

- Query on the dynamic behavior based on object entities
- Abstracting away information flow

Dynamic vs. Static Pattern

Dynamically:

```
o = req.getParameter ( );  
stmt.executeQuery (o);
```

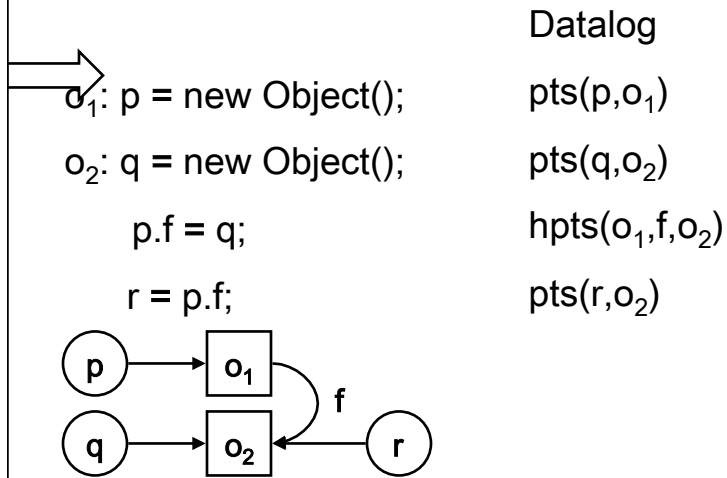
Statically:

```
p1 = req.getParameter ( );  
stmt.executeQuery (p2);
```

*p*₁ and *p*₂ point to same object?
Pointer alias analysis

Flow-Insensitive Pointer Analysis

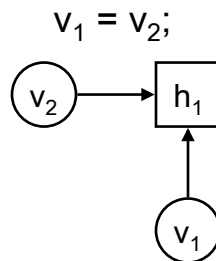
Objects allocated by same line of code are given the same name.



Inference Rule in Datalog

Assignments:

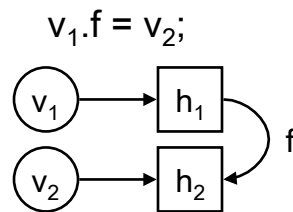
$\text{pts}(v_1, h_1) \quad :- \text{“}v_1 = v_2\text{”} \ \& \ \text{pts}(v_2, h_1).$



Inference Rule in Datalog

Stores:

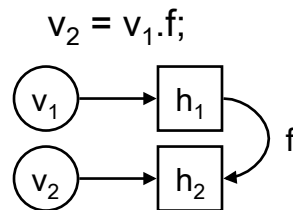
$\text{hpts}(h_1, f, h_2) \quad :- \text{“}v_1.f = v_2\text{”} \ \&$
 $\text{pts}(v_1, h_1) \ \& \ \text{pts}(v_2, h_2).$



Inference Rule in Datalog

Loads:

$\text{pts}(v_2, h_2) \quad :- \text{“}v_2 = v_1.f\text{”} \ \&$
 $\text{pts}(v_1, h_1) \ \& \ \text{hpts}(h_1, f, h_2).$



Pointer Analysis Rules

$\text{pts}(v, h) \quad :- \text{“}h: T \ v = \text{new } T()\text{”};$

$\text{pts}(v_1, h_1) \quad :- \text{“}v_1 = v_2\text{”} \ \& \ \text{pts}(v_2, h_1).$

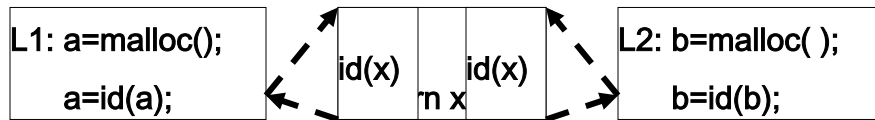
$\text{hpts}(h_1, f, h_2) \quad :- \text{“}v_1.f = v_2\text{”} \ \&$
 $\text{pts}(v_1, h_1) \ \& \ \text{pts}(v_2, h_2).$

$\text{pts}(v_2, h_2) \quad :- \text{“}v_2 = v_1.f\text{”} \ \&$
 $\text{pts}(v_1, h_1) \ \& \ \text{hpts}(h_1, f, h_2).$

Pointer Alias Analysis

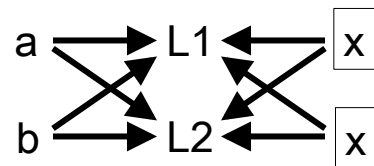
- Specified by a few Datalog rules
 - Creation sites
 - Assignments
 - Stores
 - Loads
- Apply rules until they converge

Context-Sensitive Pointer Analysis

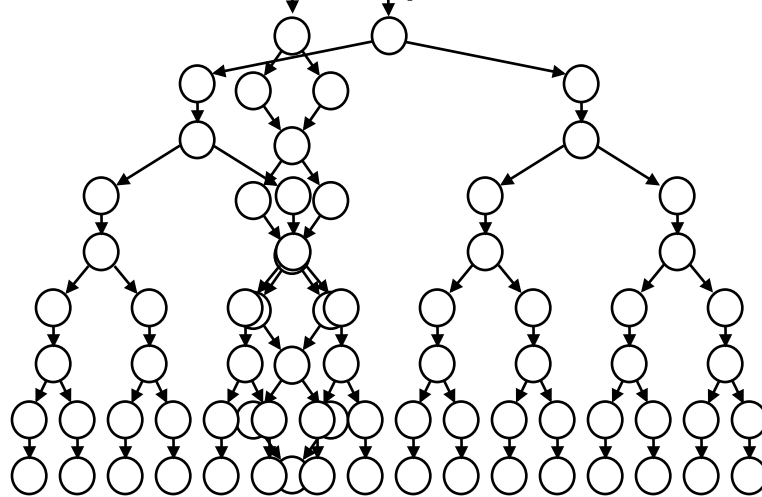


context-sensitive

context-insensitive

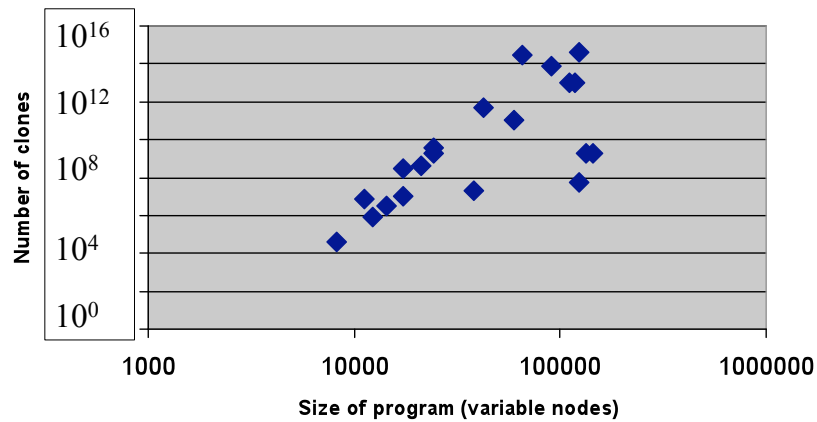


Even without recursion,
of Contexts is exponential!



Top 20 Sourceforge Java Apps

Number of Clones



Costs of Context Sensitivity

- Typical large program has $\sim 10^{14}$ paths
- If you need 1 byte to represent a context:
 - 100 terabytes of storage
 - > 12 times size of Library of Congress
 - Memory: \$1.2 million
 - Hard drive: \$47,500
 - Time to read sequentially: 20 days

Cloning-Based Algorithm

- Whaley&Lam, PLDI 2004 (best paper award)
- Create a “clone” for every context
- Apply context-insensitive algorithm to cloned call graph
- Lots of redundancy in result
- Exploit redundancy by clever use of BDDs (binary decision diagrams)

Automatic Analysis Generation



Ptr analysis in 10 lines

PQL

Datalog

bddb
(**BDD**-based
deductive **database**)
with
Active Machine Learning

1000s of lines
1 year tuning

BDD operations

BDD: 10,000s-lines library

Benchmarks

9 large, widely used applications

- Blogging/bulletin board applications
- Used at a variety of sites
- Open-source Java J2EE apps
- Available from SourceForge.net

Vulnerabilities Found

	SQL injection	HTTP splitting	Cross-site scripting	Path traversal	Total
Header	0	6	5	0	11
Parameter	6	5	0	2	13
Cookie	1	0	0	0	1
Non-Web	2	0	0	3	5
Total	9	11	5	5	30