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

Web Applications

Hacker \(\rightarrow\) Browser \(\leftrightarrow\) Web App \(\rightarrow\) Database

Evil Input

Confidential information leak
**SQL Injection Errors**

- Hacker
- Browser
- Web App
- Database

Give me Bob’s credit card #
Delete all records

---

**Happy-go-lucky SQL Query**

User supplies: *name*, *password*

Java program:

```java
String query = "SELECT UserID, Creditcard FROM CCRrec 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

<table>
<thead>
<tr>
<th>Inject Parameters</th>
<th>Exploit SQL injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden fields</td>
<td>Cross-site scripting</td>
</tr>
<tr>
<td>Headers</td>
<td>HTTP splitting</td>
</tr>
<tr>
<td>Cookie poisoning</td>
<td>Path traversal</td>
</tr>
</tbody>
</table>
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)

```java
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)

```java
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)

```java
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("\n");
query = tmp.toString();
Vector v = new Vector();
try {
    ResultSet results = statement3.executeQuery( query );
    ...
} catch (Exception e) {
```
PQL: Program Query Language

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

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

Dynamic vs. Static Pattern

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

Statically:
```
p_1 = req.getParameter ( );
stmt.executeQuery ( p_2 );
```

$p_1$ 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.

```
\[\begin{align*}
\sigma_1 &: \text{p} = \text{new Object()}; & \text{pts}(\text{p}, \text{o}_1) \\
\sigma_2 &: \text{q} = \text{new Object()}; & \text{pts}(\text{q}, \text{o}_2) \\
\text{p.f} &= \text{q}; & \text{hpts}(\text{o}_1, \text{f}, \text{o}_2) \\
\text{r} &= \text{p.f}; & \text{pts}(\text{r}, \text{o}_2)
\end{align*}\]
```

```
\text{Datalog}
```

```
\text{Objects allocated by same line of code are given the same name.}
```

---

**Inference Rule in Datalog**

Assignments:

\[
\text{pts} \left( v_1, h_1 \right) : \text{"} v_1 = v_2 \text{"} \ & \text{pts} \left( v_2, h_1 \right).
\]

```
\text{Inference Rule in Datalog}
```

```
\text{Assignments:}
```

```
\text{pts} \left( v_1, h_1 \right) : \text{"} v_1 = v_2 \text{"} \ & \text{pts} \left( v_2, h_1 \right).
```

Inference Rule in Datalog

Stores:

\[ hpts(h_1, f, h_2) \quad :- \quad "v_1.f = v_2" \quad & \quad \text{pts}(v_1, h_1) \quad & \quad \text{pts}(v_2, h_2). \]

\[ v_1.f = v_2; \]

\[ v_1 \rightarrow h_1 \]

\[ v_2 \rightarrow h_2 \]

Inference Rule in Datalog

Loads:

\[ \text{pts}(v_2, h_2) \quad :- \quad "v_2 = v_1.f" \quad & \quad \text{pts}(v_1, h_1) \quad & \quad hpts(h_1, f, h_2). \]

\[ v_2 = v_1.f; \]

\[ v_1 \rightarrow h_1 \]

\[ v_2 \rightarrow h_2 \]
**Pointer Analysis Rules**

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

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

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

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

**Pointer Alias Analysis**

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

L1: a = malloc();
    a = id(a);

L2: b = malloc();
    b = id(b);

case-sensitive

case-insensitive

Even without recursion,
# of Contexts is exponential!
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

- PQL
- Datalog
- bddbddb (BDD-based deductive database) with Active Machine Learning
- BDD: 10,000s-lines library
- 1000s of lines
- 1 year tuning
- Ptr analysis in 10 lines
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

<table>
<thead>
<tr>
<th></th>
<th>SQL injection</th>
<th>HTTP splitting</th>
<th>Cross-site scripting</th>
<th>Path traversal</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Parameter</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Cookie</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Non-Web</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td>5</td>
<td>30</td>
</tr>
</tbody>
</table>