DBTaint: Cross-Application Information Flow Tracking via Databases

Benjamin Davis
Hao Chen
University of California, Davis
Introduction

- Web services are highly attractive targets
- Over 60% of attacks target Web applications
- Over 80% of vulnerabilities found are in Web applications

(From SANS 2009 Top Cyber Security Risks)
Cross-Site Scripting

<h1>Latest Comment</h1>
<p>
{User Content}
</p>
Cross-Site Scripting

<h1>Latest Comment</h1>
<p>This is <b>great!</b></p>
Cross-Site Scripting

<h1>Latest Comment</h1>
<p>
<script>
steal(document.cookie);
</script>
</p>
Information Flow Tracking
Information Flow Tracking

Information Flow Tracking System

Application

Input
Information Flow Tracking

Information Flow Tracking System

Application
Information Flow Tracking

Information Flow Tracking System

Application
Information Flow Tracking

Information Flow Tracking System

Application

Output
Information Flow Tracking

Information Flow Tracking System

Application

Output
Examples

- Language-based “taint mode”
  - Perl
  - Ruby

- Adding support to language structures
  - Java [Chin, Wagner 09]
  - PHP [Venema]
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Input

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Input

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Database

Input

Output
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Input

Web Application

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Input

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Input

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Web Application

Database Interface

Input

Output

Database
Limitations of Single-Application Systems

Information Flow Tracking System

Database Interface

Web Application

Input

Output

Database
Limitations of Single-Application Systems

- What if you have multiple applications?
- How to treat data from the database?
  - All tainted $\rightarrow$ false positives
  - All untainted $\rightarrow$ false negatives
  - Require manual annotation?
  - Application-specific decisions?
System–Wide Approaches

- Taint tracking through the entire system
  - [Asbestos, 05]
  - [HiStar, 06]
- Implemented in
  - Hardware
  - OS
  - VMM/emulator
Process-level System-Wide Tracking
Process-level System-Wide Tracking
Process-level System-Wide Tracking

Database Interface

Web Application

Input

Output

Database
Process-level System-Wide Tracking

Diagram:
- **Web Application**
- **Database Interface**
- **Database**

Connections:
- Input from Web Application to Database Interface
- Output from Database Interface to Database
- Input from Database Interface to Web Application
- Output from Web Application to Database Interface
Low-level System-Wide Systems

- Low level/fine granularity
  - Hardware mechanism [Suh, Lee, Devadas 04]
  - Minos [Crandall, Chong, 04]
- Lacks high-level database semantics
  - Aggregate functions
  - Comparisons, SELECT DISTINCT
DBTaint Design Goals

- End-to-end taint tracking
  - Across Web applications and databases
- Leverage existing single-application information flow tracking engines
- Compatible with existing Web services
  - Require no changes to Web applications
- Taint propagation through database functions
Web Service

DB Interface

SQL

Web Application

Database Engine
DBTaint Web Service

DBTaint
DB Interface

Web Application

SQL

Database Engine

Single-application information flow
Store taint data in database composite types

- Tuple of form: (value, taint_value)

Store/retrieve taint values via SQL

- No additional mechanisms needed in the database
- No change to underlying database data structures

### Before DBTaint

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>'closed'</td>
</tr>
<tr>
<td>27</td>
<td>'open'</td>
</tr>
<tr>
<td>32</td>
<td>'pending'</td>
</tr>
</tbody>
</table>

### With DBTaint

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>('closed', 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>('open', 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>('pending', 1)</td>
</tr>
</tbody>
</table>
Create functions that operate on composite types
  ◦ Comparison operators (=, !=, <, ...)
  ◦ Arithmetic operations (+, -, ...)
  ◦ Text operations (upper, lower, ...)
  ◦ Aggregate functions (MAX, MIN, SUM, ...)

Functions implemented in SQL
  ◦ CREATE FUNCTION
  ◦ CREATE OPERATOR
  ◦ CREATE AGGREGATE
Database Taint Behavior

- Arithmetic operations

\[(4, 0) + (5, 1) = (9, ?)\]
Database Taint Behavior

- Arithmetic operations

\[(4, 0) + (5, 1) = (9, ?)\]

untainted \hspace{2cm} tainted
Database Taint Behavior

- Arithmetic operations

$$(4, 0) + (5, 1) = (9, 1)$$
Database Taint Behavior

\[
\text{MAX} \quad \{(2, 0), (3, 1), (5, 0)\} = (5, ?)
\]
Database Taint Behavior

\[ \text{MAX} \{ (2, 0), (3, 1), (5, 0) \} = (5, ?) \]
Taint Philosophy

- Untainted: trusted source
  - Web application defaults
  - Values generated entirely by the Web application
- Tainted: from untrusted source, or unknown
  - User input

- Explicit information flow
- Database returns untainted value only if database has received that value untainted
Database Taint Behavior

\[ \text{MAX} \{(2, 0), (3, 1), (5, 0)\} = (5, ?) \]
Database Taint Behavior

\[ \text{MAX} \{ (2, 0), (3, 1), (5, 0) \} = (5, 0) \]

untainted  canceled
untainted  canceled
untainted  canceled
untainted  canceled

Database Taint Behavior

- Equality

\[
(3, 0) \ ? \ (3, 1)
\]

untainted \[\text{untainted}\] \[\text{tainted}\]
Database Taint Behavior

- Equality

3 === 3
Database Taint Behavior

- Equality

\[(3, 0) == (3, 1)\]

- Adopt notion of backwards-compatibility
[Chin, Wagner 09]
Database Taint Behavior

\[\text{MAX}\{ (5, 1), (5, 0) \} = (5, \ ?)\]

- tainted
- untainted
Database Taint Behavior

- MAX

\[ \{5, 5\} = 5 \]
Database Taint Behavior

- MAX

\[
\{5, 5\} = 5
\]
Database Taint Behavior

- MAX

\{ (5, 1), (5, 0) \} = (5, ?)

OR
Database Taint Behavior

- \( \text{MAX} \)

\[ \{ (5, 1), (5, 0) \} = (5, 0) \]

- When possible, prefer to return untainted values
Web Service Information Flow

DB Interface

WebApp

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>‘closed’</td>
</tr>
<tr>
<td>27</td>
<td>‘open’</td>
</tr>
<tr>
<td>32</td>
<td>‘pending’</td>
</tr>
</tbody>
</table>
Web Service Information Flow

DB Interface

x = DB.get(id=27)

WebApp

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>‘closed’</td>
</tr>
<tr>
<td>27</td>
<td>‘open’</td>
</tr>
<tr>
<td>32</td>
<td>‘pending’</td>
</tr>
</tbody>
</table>
Web Service Information Flow

WebApp

x = DB.get(id=27)

DB Interface

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>'closed'</td>
</tr>
<tr>
<td>27</td>
<td>'open'</td>
</tr>
<tr>
<td>32</td>
<td>'pending'</td>
</tr>
</tbody>
</table>
Web Service Information Flow

DB Interface

x = DB.get(id=27)

WebApp

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>‘closed’</td>
</tr>
<tr>
<td>27</td>
<td>‘open’</td>
</tr>
<tr>
<td>32</td>
<td>‘pending’</td>
</tr>
</tbody>
</table>
Web Service Information Flow

WebApp

DB Interface

x = “open”

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>‘closed’</td>
</tr>
<tr>
<td>27</td>
<td>‘open’</td>
</tr>
<tr>
<td>32</td>
<td>‘pending’</td>
</tr>
</tbody>
</table>
DBTaint Information Flow

DBTaint

DB Interface

WebApp

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>(‘closed’, 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>(‘open’, 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>(‘pending’, 1)</td>
</tr>
</tbody>
</table>
### DBTaint Information Flow

#### DB Interface

```python
x = DB.get(id=27)
```

#### WebApp

#### Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>(‘closed’, 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>(‘open’, 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>(‘pending’, 1)</td>
</tr>
</tbody>
</table>
DBTaint Information Flow

DBTaint

DB Interface

Rewritten query

WebApp

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>(‘closed’, 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>(‘open’, 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>(‘pending’, 1)</td>
</tr>
</tbody>
</table>
DBTaint Information Flow

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>('closed', 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>('open', 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>('pending', 1)</td>
</tr>
</tbody>
</table>

WebApp

DBTaint

DB Interface

Result tuples
DBTaint Information Flow

Collapse tuples and taint appropriately

Database Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>('closed', 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>('open', 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>('pending', 1)</td>
</tr>
</tbody>
</table>
x = "open"
// x is tainted

<table>
<thead>
<tr>
<th>Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19, 0)</td>
<td>(‘closed’, 1)</td>
</tr>
<tr>
<td>(27, 0)</td>
<td>(‘open’, 1)</td>
</tr>
<tr>
<td>(32, 0)</td>
<td>(‘pending’, 1)</td>
</tr>
</tbody>
</table>
Database Client–Server Integration

- Account for composite types in SQL queries
- Collapse and taint result tuples as needed
- These changes are:
  - Transparent to web application
  - High-level, portable

[Diagram showing DBTaint, DB Interface, and DB with unchanged arrows]
Parameterized queries

Prepare:
  - INSERT ... (id, status) VALUES (?, ?)

Execute
  - (27, ‘open’)
Example: Parameterized INSERT

- Parameterized queries
- Prepare:
  - `INSERT ... (id, status) VALUES (?, ?)`
  - `// with DBTaint:`
  - `INSERT ... (id, status) VALUES (ROW(? ,?), ROW(? ,?))`
Example: Parameterized INSERT

- Parameterized queries
- Prepare:
  - `INSERT ... (id, status) VALUES (?, ?)`
  - `// with DBTaint:
    - `INSERT ... (id, status) VALUES (ROW(? ,?), ROW(? ,?))`

- Execute
  - `(27, 'open') // 27 is untainted, 'open' is tainted`
  - `// with DBTaint:
    - (27, 0, 'open', 1)`
Rewriting Parameterized Queries

- **Prepare phase:**
  - Queries are passed with placeholders for data

- **Execute phase:**
  - Data values are passed separately, independently

- **Taint tracking engine requirement:**
  - Only need to track taint values per variable

- **We handle non-parameterized queries too**
  - See paper for details
Information Flow in the Web App

- Leverage existing single-application information flow tracking systems
- No changes to Web application
Implementation

- Languages
  - Perl
  - Java

- Database Interfaces
  - Perl DataBase Interface (DBI)
  - Java Database Connectivity (JDBC)

- Database
  - PostgreSQL
Evaluation

- **RT: Request Tracker (ticket tracking system)**
  - 60,000+ lines of Perl
  - Perl DBI (DataBase Interface) API
  - Perl taint mode
- **JForum (discussion board system)**
  - 30,000+ lines of Java
  - Java Database Connectivity (JDBC) API
  - Character-level taint engine [Chin, Wagner ’09]
Performance Evaluation

![Bar chart showing requests/second for RT and JForum with Original and DBTaint categories]
Enhanced Taint Tracking

- Cross-application information flow tracking
- Persistent taint tracking
- Multiple Web applications, multiple Databases
Conclusion

- End-to-end information flow through Web services
- Compatible with existing Web services
  - Requires no changes to Web applications
- Taint propagation through database functions