Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks

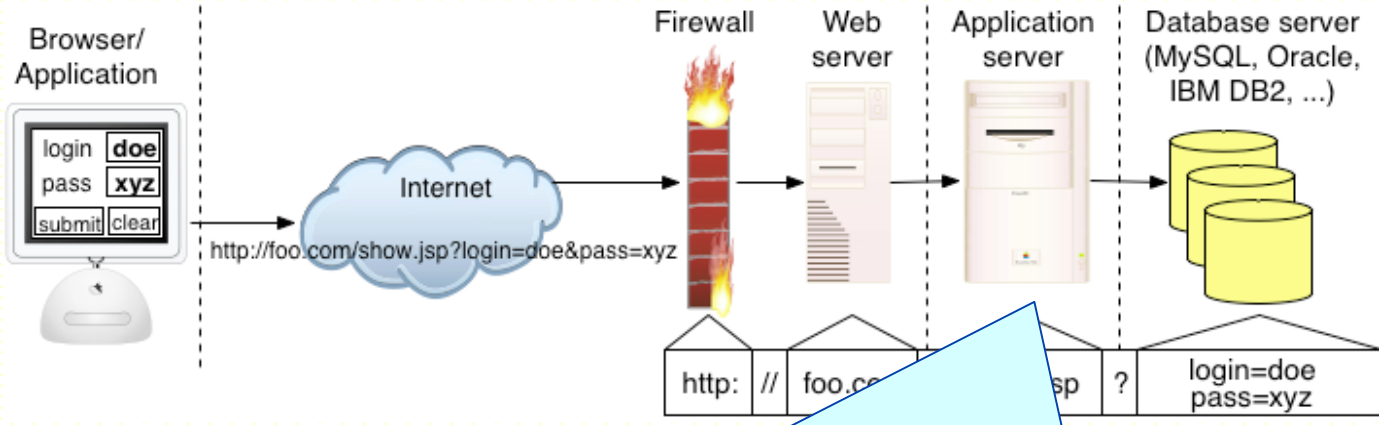
William Halfond & Alessandro Orso

Georgia Institute of Technology

This work was supported in part by NSF awards CCR-0306372, CCR-0205422, and CCR-0209322 to Georgia Tech.

SPARC

Vulnerable Application




```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals(""))) {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.executeQuery(queryString);
```

Attack Scenario

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals("")))  {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.executeQuery(queryString);
```

Normal Usage

- User submits login "**doe**" and password "**xyz**"
  - *SELECT info FROM users WHERE login='**doe**' AND pass='**xyz**'*

Attack Scenario

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals("")))  {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.executeQuery(queryString);
```

Malicious Usage

- Attacker submits "**' or 1=1 --**" and password of ""
  - *SELECT info FROM users WHERE login='* ***or 1=1 --*** *' AND pass='*

Presentation Outline

- Related Work
- Our Solution
- Implementation Details
- Preliminary Results

Related Approaches

- Program Analysis
  - Information Flow Reasoning [Huang04]
  - Type Analysis [Gould04]
  - Check for Tautologies [Wasserman04]
- Defensive Coding [WSC03]
- Proxy Filtering [Scott02]
- Randomized Instruction Set [Kc03]
- Penetration Testing [Huang03]

Our Solution

Basic Insights

1. Code contains enough information to accurately predict and model all possible queries.
2. A SQL Injection Attack will not conform to the predicted model.

Solution:

Static analysis => build query models

Runtime analysis => enforce models

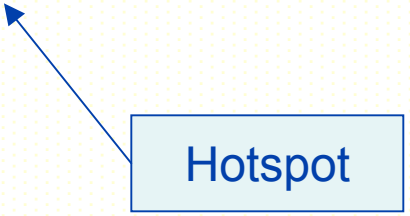
Overview of Analysis

1. Identify all hotspots.
2. Build SQL query models for each hotspot.
3. Instrument hotspots.
4. Monitor application at runtime.

1 -- Identify Hotspots

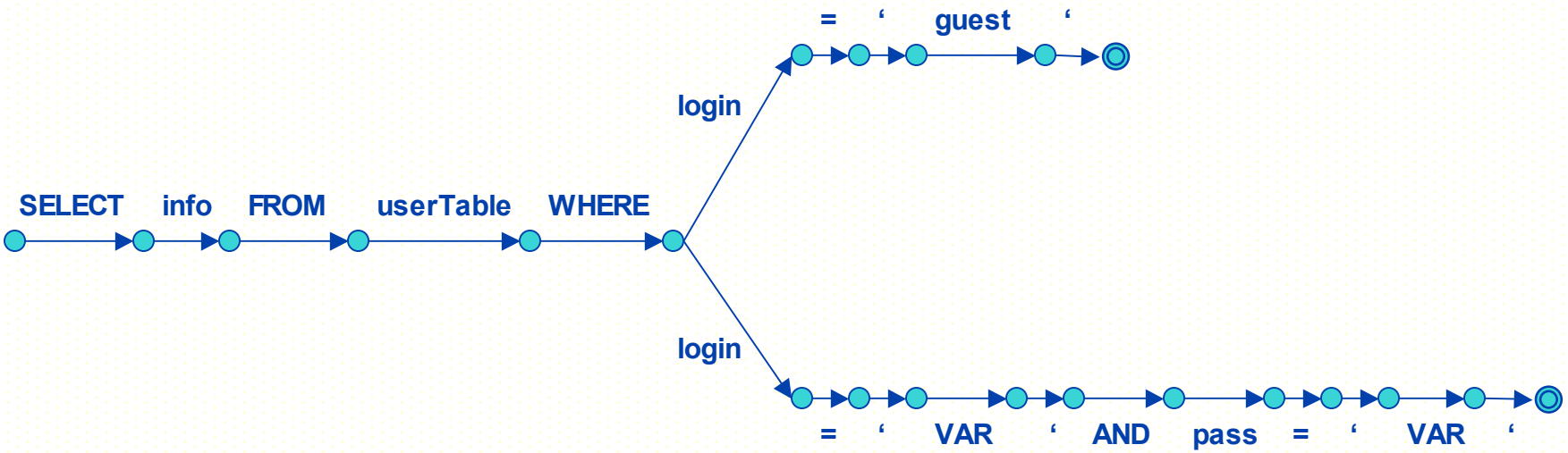
Scan application code to identify hotspots.

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals("")))  {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
ResultSet tempSet = stmt.executeQuery(queryString);
```



2 -- Build SQL Query Model

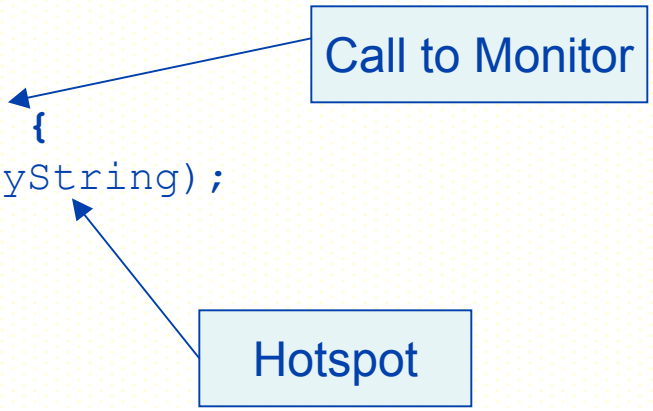
1. Use JSA [Christensen03] to construct character-level automaton.
2. Parse graph (similar to [Gould04]) to group characters into SQL tokens.



3 -- Instrument Application

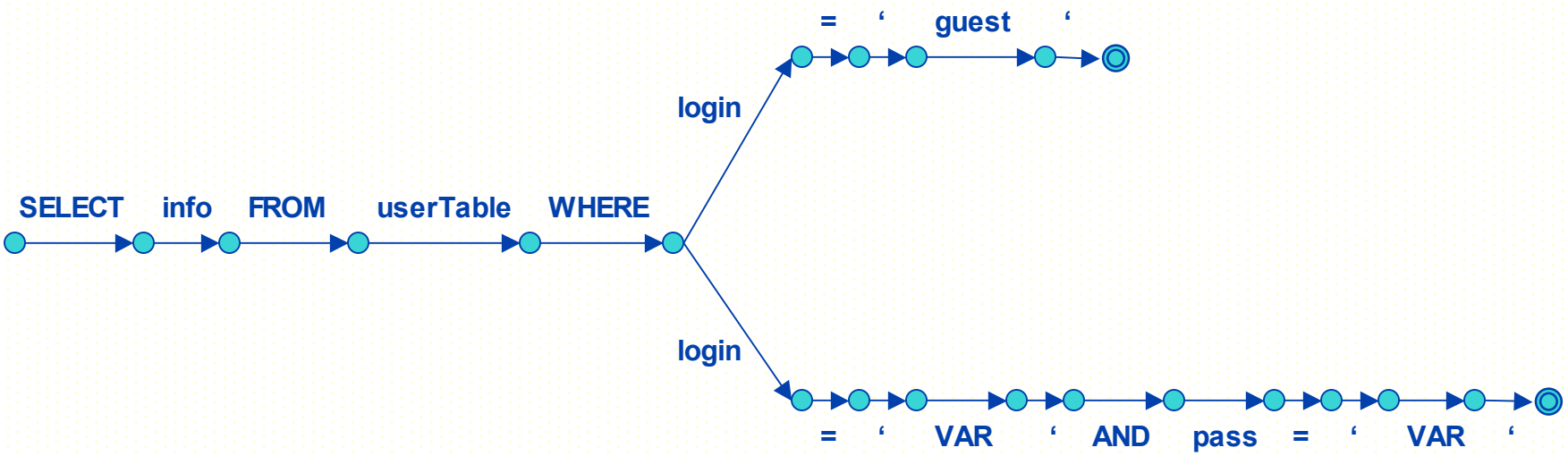
Wrap each hotspot with call to monitor.

```
String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! password.equals("")))  {
  queryString += "login='" + login + "' AND pass='" + password + "'";
} else {
  queryString+="login='guest'";
}
if (monitor.accepts (hotspotID, queryString)) {
  ResultSet tempSet = stmt.executeQuery(queryString);
}
```



4 -- Runtime Monitoring

Check queries against SQL query model.

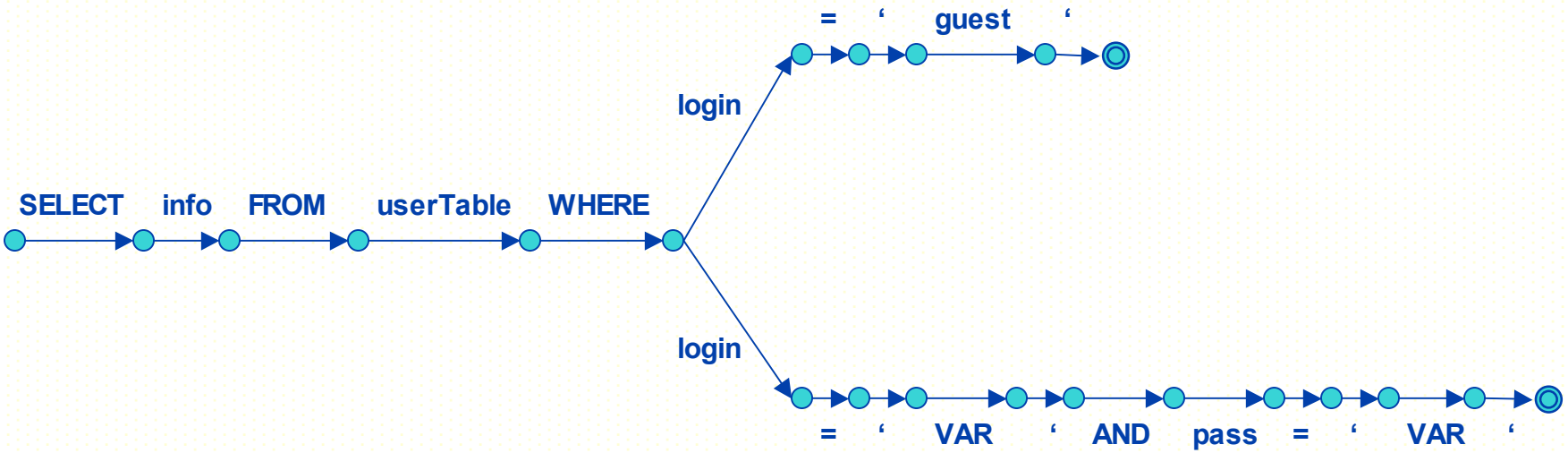


Normal Usage:

SELECT	info	FROM	userTable	WHERE	login	=	‘	doe	‘	AND	pass	=	‘	xyz	‘

4 -- Runtime Monitoring

Check queries against SQL query model.



Malicious Usage:

SELECT | info | FROM | userTable | WHERE | login | = | ‘ | ‘ | OR | 1 | = | 1 | -- | ‘ | AND | pass | = | ‘ | ‘

Implementation

Analysis Module: (Steps 1 & 2)

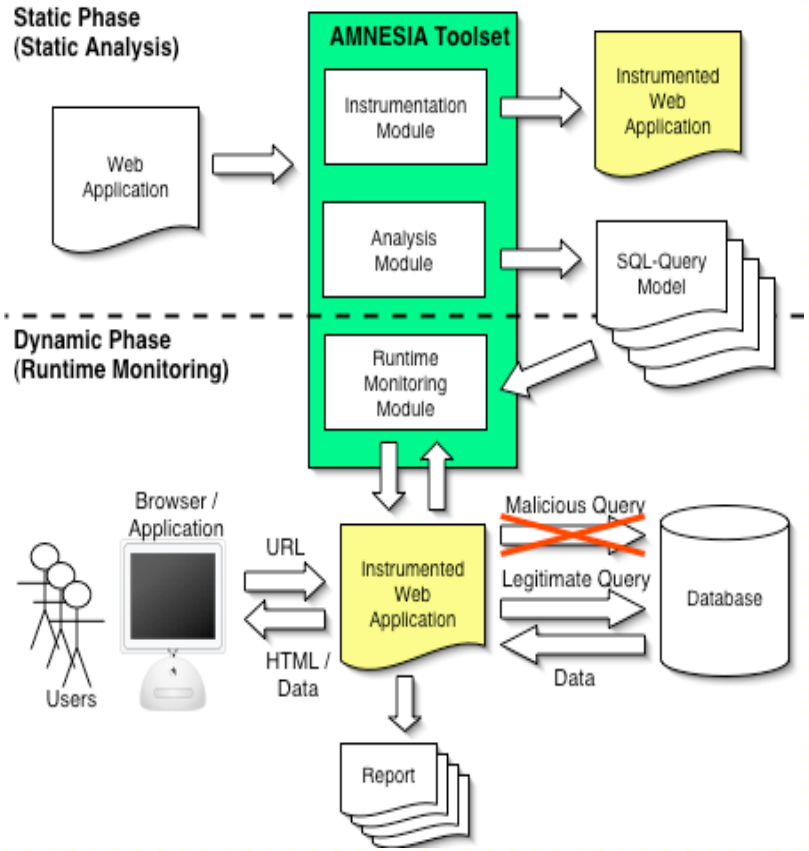
- String Analysis: JSA [Christensen03]
- SQL Tokenizing: Modified depth-first traversal

Instrumentation: (Step 3)

- InsECT [Chawla04]

Run-time Monitoring: (Step 4)

- Monitoring Library: InsECT [Chawla04]
- Runtime Checker: NDFA implementation



Preliminary Results

- Used two applications
  - Identified vulnerable hotspots
  - Crafted targeted attack queries and normal queries
  - Evaluated effectiveness of technique for protecting applications
- No false positives or negatives.

Future Work

- More extensive and realistic evaluation
- Identify limitations of analysis
- Evaluate scalability of technique
- Use of dynamic techniques to construct model where static analysis fails

Questions