Directed Model Checking of Web Applications

Michael Martin Monica Lam

Stanford University

March 20, 2006

Overview

- Motivation
- Overview of Approach
- Basic Technique
- Refined Technique
- Experimental Results

Where We Stand

- Dynamic Analysis: PQL
  - Pattern language on traces
  - Java-like syntax
  - Triggers actions on matches
- Target Domain: Java web applications
  - Online defense against some attacks
  - Detect intrusions or application errors

Sample PQL Query

```
query StringProp (object * x)
returns object * y;
matches { y.append(x) | y = x.toString(); }

query StringPropStar (object * x)
returns object * y;
uses object * temp;
matches {
    y := x
  | { temp := StringProp(x); y := StringPropStar(temp); }
}

query main ()
returns object String source, tainted;
matches {
    source = javax.servlet.http.HttpServlet.getParameter();
    tainted := StringPropStar(source);
    java.sql.Connection.prepareStatement(tainted);
}
```

Online Isn’t Good Enough

- Some problems can’t be fixed online
- Catching a match won’t tell why
- Not systematic
- Overhead is a continuing cost

Catching everything ahead of time is better

Systematic Testing

- Simple execution model
  - String comes in (URL)
  - String goes out (Web page)
  - Repeat
- Application state mutable by requests
  - Typically per-user, occasionally global
- Problem is *input generation*
  - Find URL sequences that excercise app
  - URLs in isolation are nice but not sufficient

Overview

- Motivation
- Overview of Approach
- Basic Technique
- Refined Technique
- Experimental Results

Input Generation

- Surprisingly feasible for Java apps
  - Java webapps self-document
  - “Servlet container” parses the URL
  - We generate the parsed data, not URLs
- Simulate databases and rest of backend
- Produces a self-contained application

Model Checking

- Apply dynamic instrumentation to app
- Model check complete package
  - PQL match is just part of the program
- Millions of possible requests
- Solution: Guide the checker
  - PQL Query informs static analysis
  - Analysis results give priorities for inputs

Experimental Results

- Proof of Feasibility
  - Duplicated dynamic results from initial work with PQL
  - Dynamically triggered bugs only static found previously
- Found new bugs
  - Improved harness found additional injection vectors
  - Static heuristics moved matches
- Cross-request Analysis
  - Force logins, handle redirects
  - One experiment needed this to run at all

Overview

- Motivation
- Overview of Approach
- Basic Technique
- Refined Technique
- Experimental Results

Building a Basic Harness

- Java Servlets self-document
- **`web.xml`** specifies all entry points
  - **`servlet-class: doGet(), doPost()`**
  - **`filter-class`**
  - **`listener`**
- User input is handled purely via the **`HttpServletRequest`** class
- Handled with reflection “in the wild”
  - Hardcoded in harness

Building a Basic Harness

- Other frameworks build on Servlets
- Apache Struts is a popular MVC framework for this purpose
- Only one servlet, which dispatches to **`Actions`**
- User input is preconstrained to fit into **`ActionForms`**

Modeling the Environment

- Randomly select entry points
  - Each is one URL
  - Web page layout is and *must be* ignored
- Randomly fill in user input
  - Pool of possible responses
  - Currently hand-generated
    - » numbers
    - » booleans
    - » General strings
  - Select values lazily

Running the Dynamic Analysis

- Online analyses just work
  - Checker does backtracking
  - Checker does resource management
- File access not allowed
  - Hardcode data from analysis config
- PQL dynamic works nearly unchanged
  - Query compiled into static initializer
  - Signal model checker on match

Running the Model Checker

- Java Pathfinder is straightforward
- However, too many combinations
- Complete check: 10-15 hours
- Matches fall into two categories:
  - Rare
  - Nearly universal
- Checking stops on match or error

Controlling the Model Checker

- Keep log of random decisions
- Force backtracks on:
  - Paths checked in previous run
  - Uninteresting error
- Choose selection order
  - Give priority to “interesting” entry points
  - Static analysis to find interesting points
  - Various heuristics based on PQL query

Overview

- Motivation
- Overview of Approach
- Basic Technique
- Refined Technique
- Experimental Results

Simplest Heuristic

- Centers on “final events”
  - A *final event* completes a PQL match
- No request lacking final events is interesting
- Call graph analysis
  - Credit each final event to any entry point that can call it
- Priority to actions with most final events

Full-Query Heuristic

- Check for matches of the entire query
- Full context-sensitive analysis
- Requests can interfere
  - Solution: Individual harnesses for actions
- Sort by:
  - Relevant program points
  - Number of possible combinations

Find Matches Fast

- We want to optimize matches over *time*
- Model checker is depth-first
  - Actions are completely exhausted
  - Test cases grow exponentially
- Get small actions out of the way first
  - 2 parameters: < 5 seconds to search
  - Many actions have > 10 parameters
- May conflict with prior heuristics

Finding Cross-Request Matches

- Naïve approach:
  - All request chains of length 1
  - All request chains of length 2
  - All request chains of length 3
  - ...
  - Repeat until patience runs out
- Patience runs out at “chains of length 1”

Heuristics Sort of Work

- Simple final-event heuristic helps a bit
  - Only constrains the last request
- Full-Query Heuristic helps more
  - “Individual harnesses” built for sequences
- Both too coarse
  - Ignore that HTTP is stateless
- Must track *information flow* across requests

Persistent State in Servlets

- The **`HttpSession`** class
  - Simple key-value mapping
  - Per-user
  - Persists across user-requests
- Servlet fields
  - Servlets are singletons
  - Mutable servlet fields are possible
    - » Highly deprecated
- Databases, Filesystems, etc.

Dependencies

- Two web requests A and B
- A *may depend on* B if:
  - B writes a value v to a key k in its session
  - A reads from key k in its session
- Only check sequences where:
  - For every request R, some subsequent request may depend on R
  - Final request passes earlier heuristics

Finding Dependencies

- This is surprisingly feasible statically
- Keys are almost always constant strings
  - Often, static final fields
  - Results immediate from pointer analysis
- Approximate soundly
  - Non-constants can be anything
  - Didn’t come up in our experiments

Overview

- Motivation
- Overview of Approach
- Basic Technique
- Refined Technique
- Experimental Results

Experimental Topics

- Revisit an old application
  - More static bugs than dynamic
  - Use model checking to close the gap
- Analyze new applications
  - Search for unknown bugs
- Test optimization heuristics

Experimental Results

Application	Injs	Actions	Simple	Full	Chains
personalblog	3	15	3	2	0
jgossip	0	80	71	0	410
jorganizer	8	46	31	18	96

Legacy Case: personalblog

- Appeared in OOPSLA’05 PQL paper
  - 2 possible SQL injections found statically
  - Only 1 dynamically confirmed
- Built a new harness, model-checked
  - Found both static cases dynamically
  - Resolving `ActionForm` reflection discovered a third injection
- Many unchecked exceptions from invalid input

personalblog: Heuristics

- Basic heuristic extremely effective
  - Top two actions to test contained all three vulnerabilities
  - No actions actually eliminated
- Full-query heuristic restricts results to just the two vulnerable actions
- No cross-request vulnerabilities found

New case: jgossip

- Simple heuristics do not reject anything
- No injections found
- Nearly all SQL from string constants
- Exception passed through a sanitizer
  - Searched for non-constant query string
  - Code inspection on sanitizer looked OK
- Strong evidence code is clean

New case: jorganizer

- Had many traditional injections
- None reachable if Session data wrong
- Request analysis works this out

Related Work

- Model Checkers
  - SPIN, Bandera, CMC, JPF
- Model Checkers as bug finders
  - FiSC, WebSSARI
- Bug Finders
  - Metal, Partiqle, PREfix, Clouseau
- Input Generation
  - Korat, DART, Cadar

Conclusions

- Model Checking servlets is feasible
  - Finds bugs
  - Servlets are well-documented
- Multirequest tracking is important
  - Static analysis tracks important cases
- Tightly bound hybrid analysis
  - Static harness directly models environment
  - Dynamic lists out all possible flow