LAPSE: a Security Auditing Tool for Java

Benjamin Livshits

Stanford University, Computer Systems Lab

Introduction

- Security errors are common in today's Java programs
- Lead to stolen or corrupt data, system downtime
- 92% of Web apps are vulnerable to attack [Imperva]
- Recent **kinds of security attacks** appeared
  - **Parameter manipulation**
  - **Header manipulation**
  - **Cookie poisoning**
  - **Command-line params**
  - **SQL injections**
  - **Cross-site scripting**
  - **HTTP splitting**
  - **Path traversal**
- What do we do? How do we protect our applications?
- How do we prevent these vulnerabilities?
- Our approach – tool called **LAPSE**
  - **L**ightweight **A**nalysis for **P**rogram **S**ecurity in **E**clipse
  - Find the errors in the Java source code
  - Give the developer an automatic security auditing tool

Vulnerability Example: SQL injection

- Construct SQL queries based on user-provided input

```
SELECT UserID, Creditcard
    FROM Records
    WHERE
    Name = ' + name + ';
```

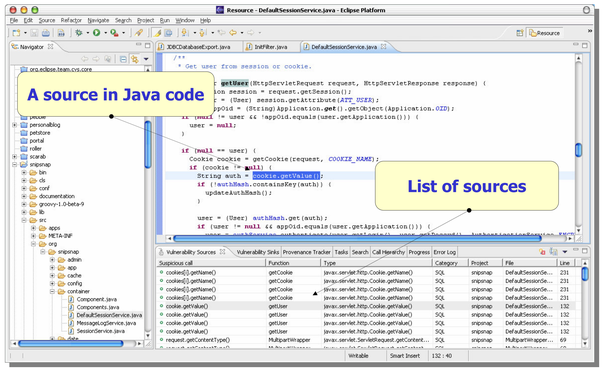
- If ***name*** is user-controlled – **danger, danger!!**

Set `name` to	Resulting SQL
`bob`	`WHERE = 'bob'`
`bob' --`	`WHERE = 'bob' -- '`
`bob' or 1=1 --`	`WHERE bob' or 1=1 -- '`
`bob'; DROP Records; --`	`WHERE bob'; DROP Records; -- '`

- How bad is this? Causes
  - Unauthorized information access
  - Deleted records

LAPSE: a Security Auditing Tool for Eclipse

- Taint problems–like taint mode in Perl, but static
- Unchecked input propagates to sensitive methods in the program
  - Sources – data enters the Web app
  - Sinks – SQL execution statements, send data back to the user, file access operations, etc.



Sources and Sinks

- Form parameters
- HTTP headers
- Cookie values
- Other types

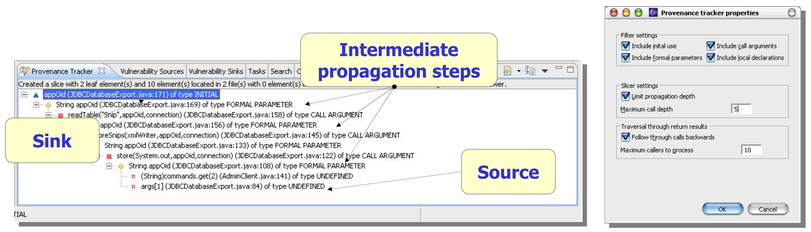
⇨

- SQL execute calls
- Output statements
- Redirect calls
- File access routines

Tracking Flow of Data between a Source and a Sink in LAPSE

1. Start at a **sink**
   - **Propagate** backwards
   - Can any **source** reach this sink?
2. Follow values through
   - Method parameters/return values
   - Local variables
   - String concatenation
3. Filter results
   - For speed
   - Not in source

- To analyze if a sink can is "dangerous" need to determine what can flow to it
  - Eclipse already allows to look up definitions of variables
  - We take this further:
    - Trace values backwards through parameters, assignments, function calls
  - If we encounter a source: stop, declare victory



Results

- Found 18 verified security errors
- In 15 open-source Web apps from SourceForge
- Most are blogging, bulletin-board programs
- Widely used and deployed at many sites
- Contain a total of
  - 2,383 classes
  - Over 524,000 of code
- Auditing of 15 apps takes under an hour

Discussion

- Auditing is pretty effective, however
  - Requires some manual effort
  - Not a complete solution – may miss errors
  - Some errors are hard to analyze
    - Sources and sinks are far apart
    - Often no source code available – only byte code
- Working on a complete solution
  - Submitted a paper to Usenix Security 2005
  - Based on a heavy-weight sound static analysis
    - Pointer analysis
    - Sound – guaranteed to find all potential errors
    - Much longer analysis times
- Working on a runtime protection solution
  - Detect errors at runtime
  - Cleanse the tainted values and proceed

References

- Security bugs in C (buffer overruns, format strings)
  - **Static**: LCLint, ITS4, Flawnder, Rats, Splint, BOON
  - **Dynamic**: StackGuard, CRED
- To the best of our knowledge, we are the 1st publically annonced Java code auditing security tool