ORACLE®

ORACLE®

Finding Security Bugs in Java Programs using Datalog

Bernhard Scholz
Nicholas Allen
Padmanabhan Krishnan

Oracle Labs, Brisbane, Australia

Disclaimer

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Program Agenda

- Java Security Issues
- Example: Caller Sensitive Methods
- Rapid Prototyping of Program Analyses in Datalog
- Security Analysis for Caller Sensitive Methods
- Experiments

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Zero-day Vulnerability Market [1]

Large Ecosystem

- Market for buying exploits
- Multi-billion dollar industry
- "Write once, run anywhere"
- Java is platform independent

Software	Estimates in USD
Adobe Reader	$5,000 - $30,000
MAC OS X	$20,000 - $50,000
Flash or Java Browser Plug-Ins	$30,000 - $60,000
Microsoft Word	$40,000 - $100,000
Windows	$50,000 - $100,000
Firefox / Safari	$60,000 - $120,000
Chrome or Internet Explorer	$80,000 - $200,000
IOS	$100,000 - $250,000

[1] http://www.net-security.org/secworld.php?id=12652 (March 2012)

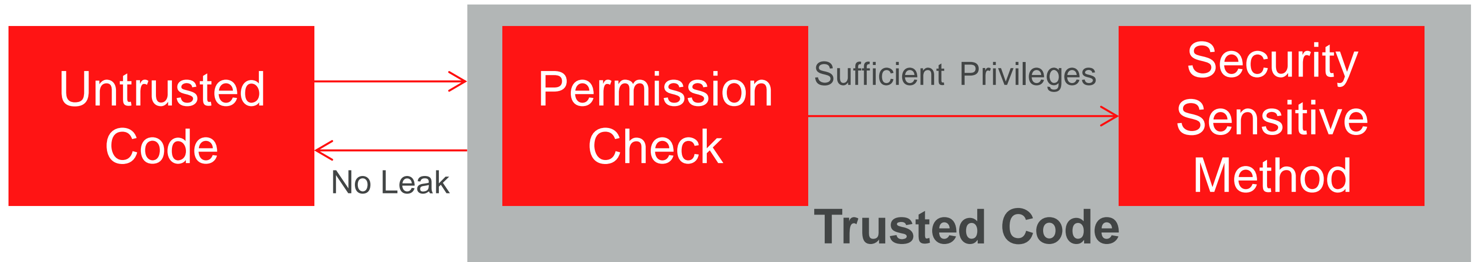
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Caller-Sensitive Methods (CSM)

One Possible Attack Vector for Java Exploits

- Security sensitive methods
  - must not be invoked unchecked on behalf of untrusted code
  - must not escape sensitive information
- If untrusted code invokes security sensitive methods
  - perform checks and prevent information leaks of sensitive information



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.
ORACLE

Caller-Sensitive Methods (CSM)

Features and Issues

- 80% of JDK's public interfaces may directly or indirectly invoke a CSM
- Example of a CSM
  - `Class c = Class.forName("sun…")`
- CSM use reflection
  - hard to analyse
- Listed in Secure Coding Guidelines
  - Access Control / Section 9
- CSM use caller's class-loader or package access capabilities

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Zero-day Exploit Example: CVE-2012-4681

Public method in **sun.awt.Toolkit**

```
public static Field getField(final Class klass,
                             final String fieldName) {
  return AccessController.doPrivileged(
     new PrivilegedAction<Field>() {
       public Field run() {
         try {
           Field field = klass.getDeclaredField(fieldName);
           field.setAccessible(true);
           return field; …
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Finding Security Bugs

Automated Tools

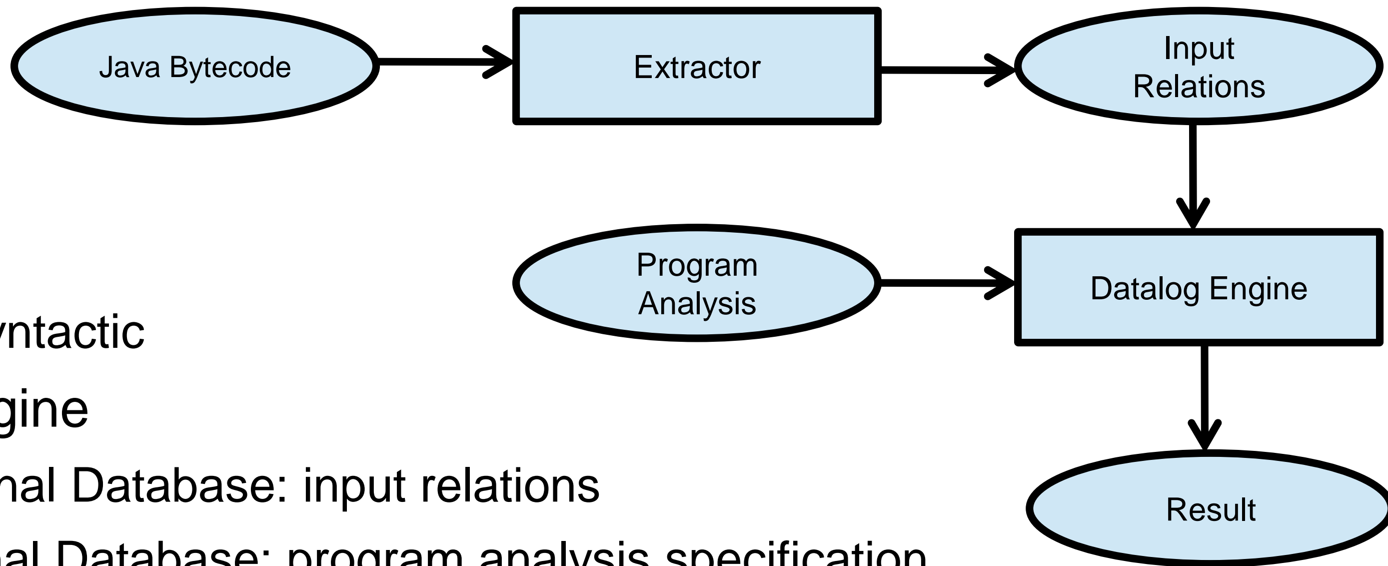
- Testing for Security Bugs
  - Testing checks functional requirements and ***not security*!**
  - Code-inspections are insufficient for finding (most) security bugs
  - Complex because of reflection, e.g., CSM
- Automated Bug-Checking Tools
  - Find security problems with static program analysis
  - By over-approximation using abstract interpretation
- Zero-day exploits ***demand rapid-prototyping capabilities***
  - Add new program analysis ***swiftly*** for new 0-day exploits

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Bug Checker in Datalog

Framework



- Extractor
  - Purely syntactic
- Datalog Engine
  - Extensional Database: input relations
  - Intensional Database: program analysis specification

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Security Analysis for CSM

Conditions

- Some conditions for causing security defects
  - Tainted inputs
    - User controls actual parameters of CSM
  - No permissions checks on a path from a public interface to CSM
  - Leak of sensitive information
- Building a security analyses in Datalog
  1. Points-to analysis
  2. Taint analysis (based on points-to analysis)
  3. All-path permission check
  4. Escape analysis (based on points-to analysis)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Points-To Analysis

Using Datalog

- Flow-insensitive, inclusion-based, context-insensitive (cf. J. Whalley'04)
- Abstract Domain
  - Variables
    - Local, actual/formal parameters, return-values, bases, this-variables
  - Heap-allocated objects
    - Creation-site as an abstraction for dynamically created objects
    - Heap-allocated object have fields
- Relations for computing points-to analysis
  - vP(v,h): variable v may point to heap object h
  - $hP(h_1,f,h_2)$: field f of h_1 may point to h_2

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Taint Analysis for CSM

Using Datalog

- Taint analysis tracks values emanating from tainted sources
  - Tainted values might be controlled by attacker
  - Tainted CSM parameters can be dangerous
- Taint analysis
  - Context- and flow-insensitive but object-sensitive
  - Public interfaces are a tainted source
  - Propagation rules for tainting objects
- Relations for computing taint analysis
  - tH(h): heap object h might be tainted
  - tV(v): variable v might be tainted

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

All-Path Permission Check

CSM

- On all paths from a public interface to a CSM callsite
  - A permission check must be performed (e.g. checkPackageAccess)
- CSM call-site could be exploited if,
  - a permission check is not performed on all paths, and
  - CSM parameters are tainted.
- Testing for all-path permission check
  - Classical dataflow analysis problem (e.g. GEN/KILL)
  - How to implement a dataflow analysis problem in Datalog?

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

All-Path Permission Check

Using Datalog

- The all-path permission check for a CSM call-site

  $\textbf{CheckedPaths(u)} \Leftrightarrow \forall \pi \in \textbf{Path(s,u)}: \exists v \in \pi: \textbf{Check(v)}$

  - **s** is a public interface
  - **u** is a statement (including CSM call-sites)
  - **Path(s,u)** is the set of all program path from **s** to **u**
  - **Check(v)** holds if statement v performs a permission check
- Dual logic because of Datalog semantics

  $\textbf{UncheckedPath(u)} \Leftrightarrow \exists \pi \in \textbf{Path(s,u)}: \forall v \in \pi: \neg \textbf{Check(v)}$

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Experiments

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.
ORACLE

Experiment

Problem Size

- OpenJDK 1.7
  - Number of variables: ≈1.5M
  - Number of heap objects: ≈400K
  - Number of methods: ≈170K
  - Number of invocations: ≈600K
  - Number of types: ≈18K

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Preliminary Results

Runtime & Effectiveness

Analyses	Time Taken
Basic (No Handling of Virtual Dispatch)	40 minutes
Virtual Dispatch + Call Graph Construction	7 hours

Intel i5-3320 (2.6GHz) machine with 16G memory running Ubuntu 12.10 using the BDDBDDB engine

	Precision (%)	Recall (%)
First Taint Model	81	94
Second Taint Model	93	80

Using a reference implementation

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Summary & Conclusion

Finding Security Bugs using Datalog

- Static program analysis is essential for checking security properties
- Implementation of program analysis using Datalog:
  - Rapid prototyping of different models
  - Extensible
  - Program analyses in Datalog are concise (=fewer bugs)
  - Debugging infrastructure still in its infancy
- Preliminary experiments
  - Datalog is efficient enough and effective

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®