ORACLE®

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 © 2017, Oracle and/or its affiliates. All rights reserved.

Inference of Security-Sensitive Entities in Libraries

Yi Lu
Oracle Labs Australia

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Access Control in JDK*

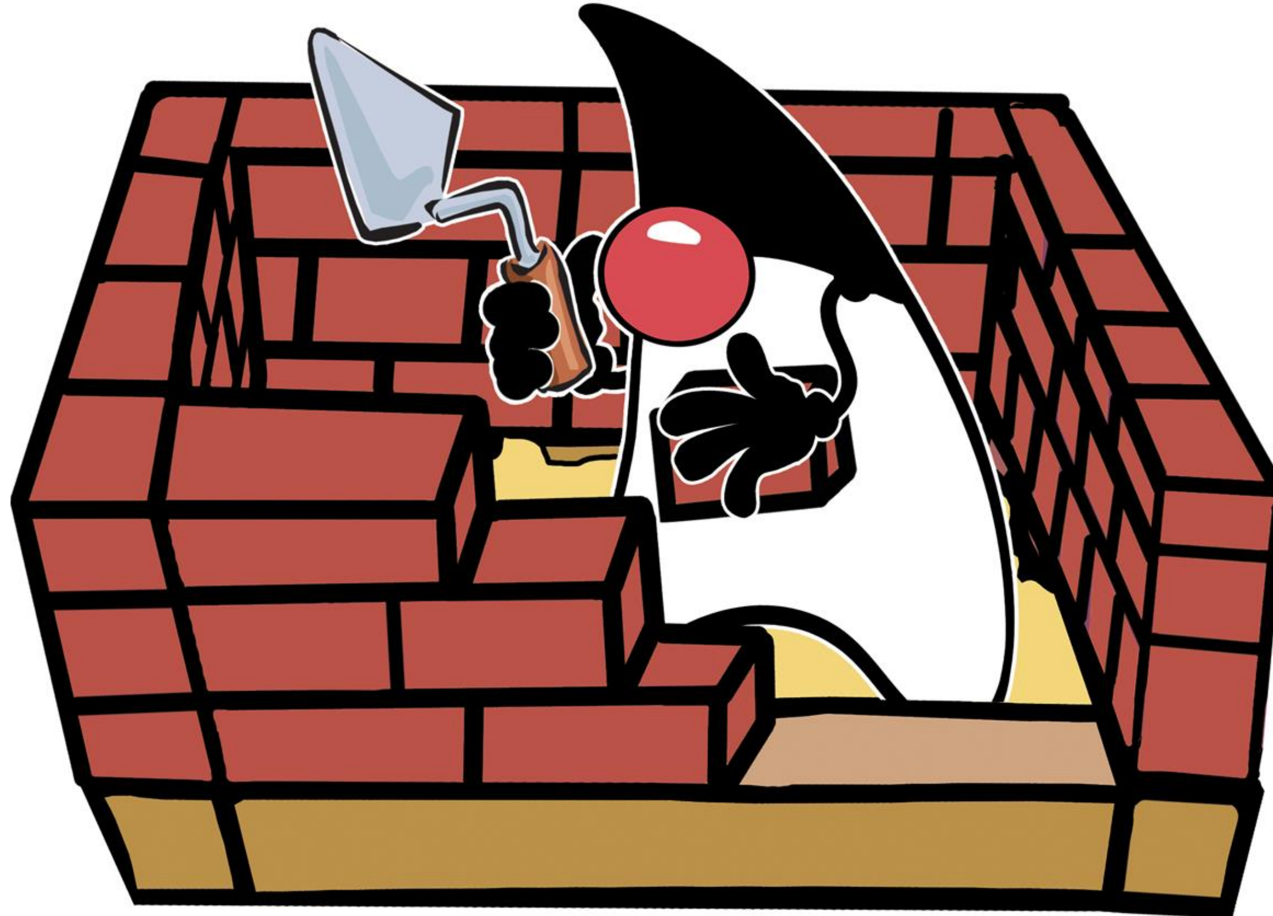
```
public final class Class<T> … {
  …
  public java.security.ProtectionDomain getProtectionDomain() {
    SecurityManager sm = System.getSecurityManager();
    if (sm != null) {
      sm.checkPermission(new RuntimePermission("getProtectionDomain"));
    }
    java.security.ProtectionDomain pd = getProtectionDomain0();
    …
    return pd;
   }

  private native java.security.ProtectionDomain getProtectionDomain0();
}
```

* Java, JDK and JRE are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Defend by Carefully Handcrafted Sandbox



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Java’s Security Model Overview

- Libraries encapsulate security-sensitive entities
  - Prevent unauthorised access by using fine-grained permissions
- Relies on programmer discipline
  - Cannot defend against implementation bugs
- How do we guarantee security?
  - Ideally by automatic tools

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Security-Sensitive Entities

- Can directly access low-level system resources
- Not specified or documented in JDK
- Important for manual and automatic analyses of security-related bugs
  - E.g. to detect unauthorised accesses, information leaks, tainted sensitive operations

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Inference of Security Specification

- *Goal*: recover security design decision of JDK to help reason about security bugs
- *Intuition*: security-sensitive entities are consistently guarded by specific permissions from public accesses
- *Approach*: analyse control dependency on permission checks

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Inference Algorithm Overview

- Candidates are selected from *publicly inaccessible* methods in JDK
- From call sites of each candidate, perform backward control dependence analysis to find previous permission checks
- A candidate is security-sensitive if *consistently* guarded by a permission

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Challenges in Inferring Security Specification for JDK

- No standard definition of security-sensitive entities
- Security-sensitive entities are not always guarded
- Permission checks are not lexically scoped
- Permission objects may contain results of some computations

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

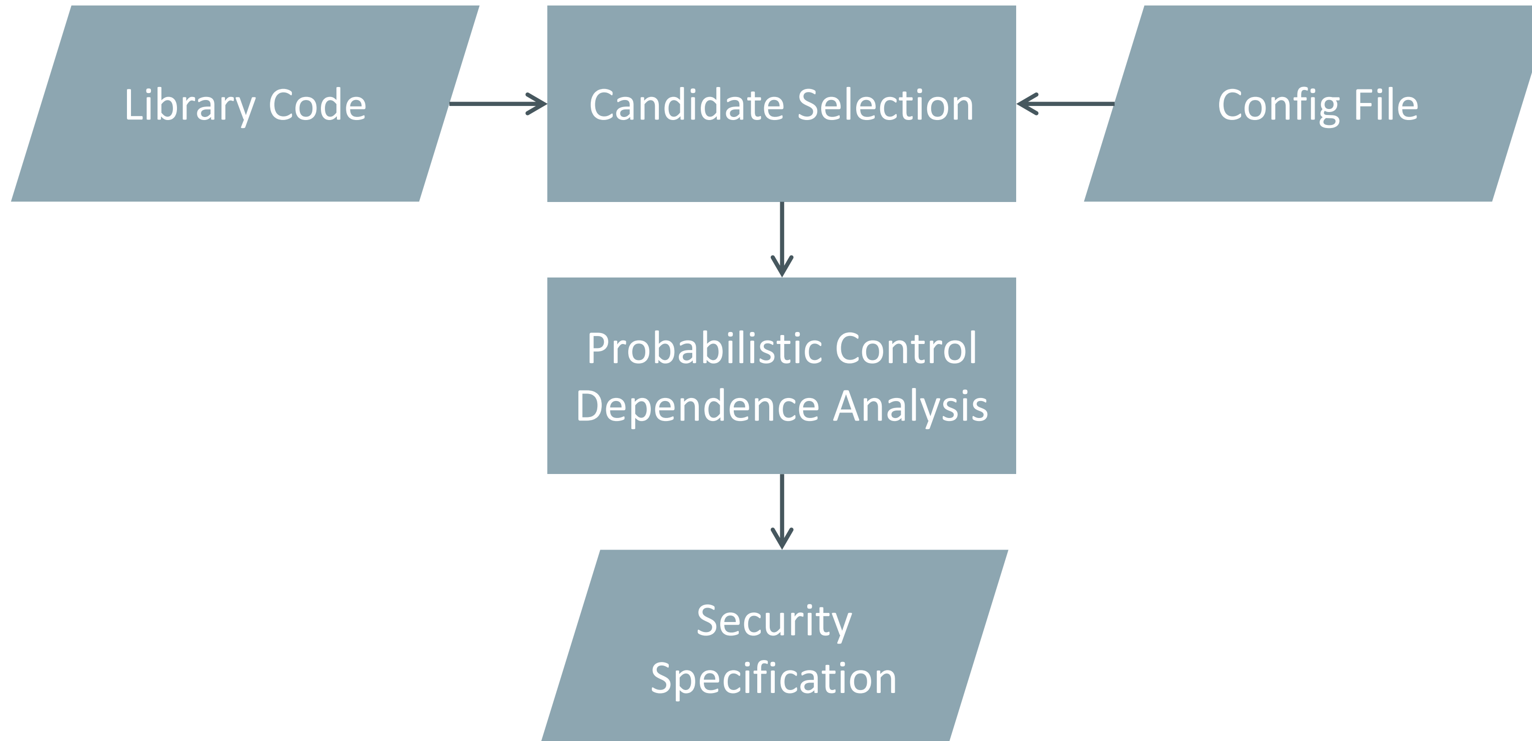
Challenges in Inferring Security Specification for JDK

- No standard definition of security-sensitive entities
  - Interprecedual control dependence analysis on permission checks
- Security-sensitive entities are not always guarded
  - Probabilistic coverage analysis to tolerate inconsistency
- Permission checks are not lexically scoped
  - Probabilistic proximity analysis to find more relevant permission checks
- Permission objects may contain results of some computations
  - Data flow analysis to resolve permission objects

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

High-level Work Flow



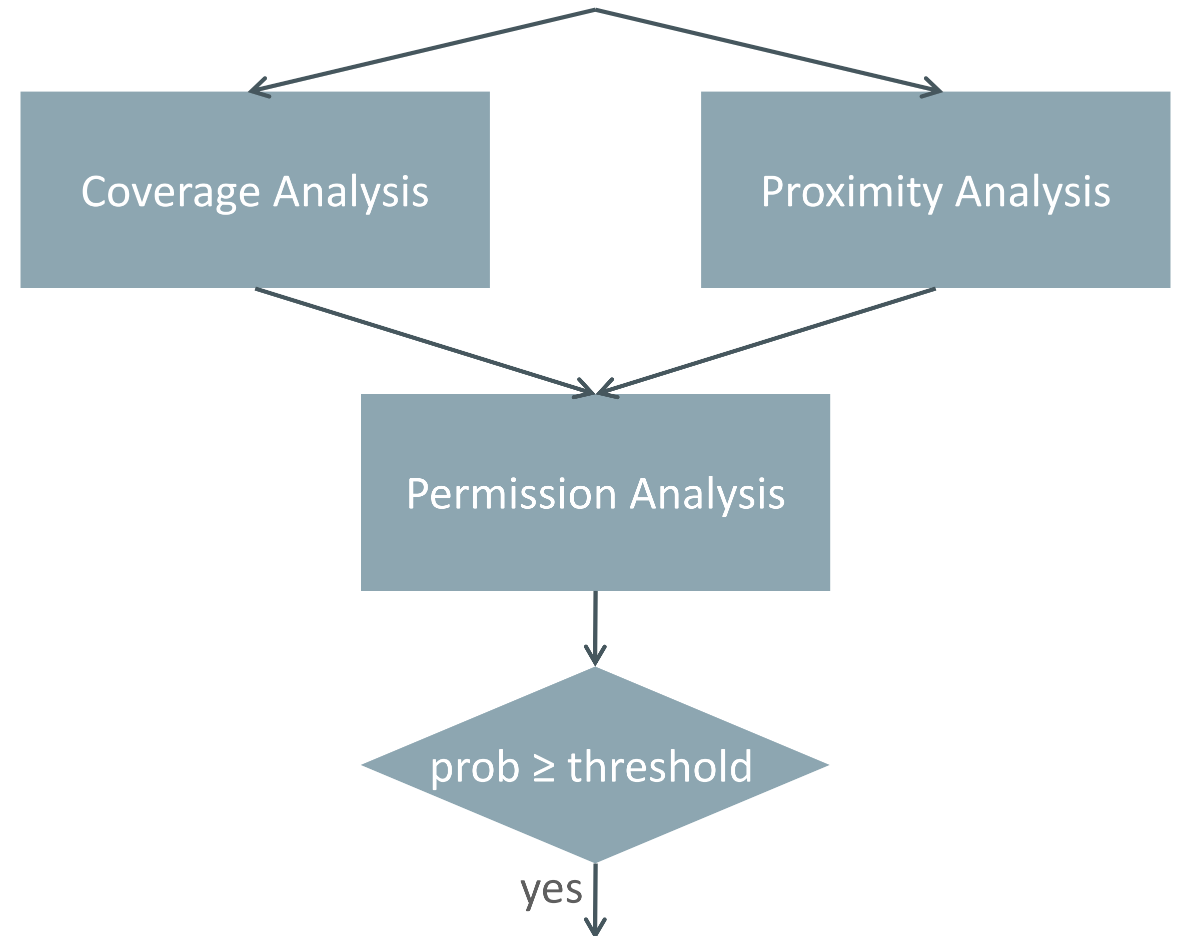
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Probabilistic Control Dependence Analysis

- Finds permission checks guarding a candidate
  - Follow control flow graphs and call graph backward till reaching public entries

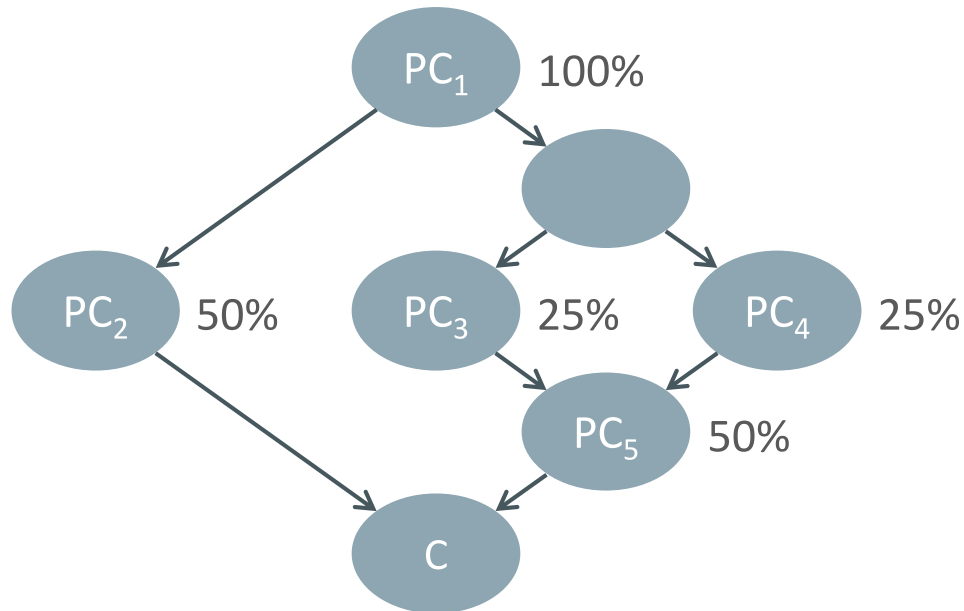
- Compute the *probability* of permission checks based on branches in control flow graphs



ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Probabilistic Control Dependence Analysis

- Coverage analysis
  - How likely candidate (C) is guarded by permission check (PC)



- Proximity analysis
  - How close are between candidate and permission check in the call graph
    - Reduce undesired shadow of non-lexically scoped permission checks

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Permission Analysis

- Resolves permissions from permission checks
  - Limited alias reasoning in the implementation due to the lack of points-to analysis in the analysis framework

- Computes the probability for permissions
  - Probabilities of different permission checks are aggregated for the same permission
  - A threshold is used to determine the relevance of the permissions

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Experiment

- Framework: Oracle Parfait
- Benchmark: OpenJDK7-b147
- OS: Solaris
- Hardware
  - CPU speed: 2993MHz
  - Core: 8
  - RAM: 131072 Mb

- The analysis takes average ~40 minutes to run

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Experimental Results

- Choice of thresholds

Inferred SSEs (*False Positives*)		Proximity	
		≤ 2	≤ 3
Coverage	≥ 0.5	109 (*28*)	238 (*117*)
	≥ 0.75	86 (*22*)	187 (*86*)
	≥ 1	83 (*20*)	173 (*77*)

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Sample Inferred Security Specification

Coverage >= 0.5 and proximity ≤ 2

Security-Sensitive Entity	Inferred Permission	Coverage	Proximity
getProtectionDomain0()	RuntimePermission (“getProtectionDomain”)	1	1
open(String name)	FilePermission(name, “read”)	1	1
delete(File f)	FilePermission(f.path, “delete”)	1	1
RandomAccessFile. open(String name, int mode)	FilePermission(name, “write”)	0.5	1
	FilePermission(name, “read”)	1	1
receive(…)	SocketPermission(?, “accpet”)	0.667	1

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Sources of False Reports

- False positives
  - Undesired shadow of non-lexically scoped permission checks
  - Conservative aggregation of probabilities of irresolvable permissions

- False negatives
  - Incomplete call graph in the framework
  - Some permission checks depend on complex semantics

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Sample False Report: State-Dependent Permission Checks

```
public class DatagramSocket implements java.io.Closeable {
   int connectState = ST_NOT_CONNECTED;

   public void connect(SocketAddress addr) … {
      …
      AccessController.checkPermission(
         new SocketPermission(host, "accept"));
       …
      connectState = ST_CONNECTED;
   }
   public synchronized void receive(DatagramPacket p) … {
      …
      if (connectState == ST_NOT_CONNECTED) {
         AccessController.checkPermission(
            new SocketPermission(host, "accept"));
      }
      …
      getImpl().receive(p);
}
```

```
abstract class AbstractPlainDatagramSocketImpl
                                        extends DatagramSocketImpl {
   …
   protected synchronized void receive(DatagramPacket p)
                                        throws IOException {
      receive0(p);
   }
   protected abstract void receive0(DatagramPacket p)
                                        throws IOException;
}

Class PlainDatagramSocketImpl
                              extends AbstractPlainDatagramSocketImpl {
   …
   protected synchronized native void receive0 (DatagramPacket p)
                                        throws IOException;
}
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Conclusion

- Security of software libraries often rely on programmer discipline

- Tools for precisely identifying security-related bugs must understand the security design intent of libraries

- Security specification of libraries may be inferred by static program analysis on security checks used in libraries

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Q&A

ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Integrated Cloud

Applications & Platform Services

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®