ORACLE®

Safe Harbor Statement

The following is intended to outline our general product direction. 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. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

From Access Control to Information Flow

Security Model and Static Analysis for Programs Consist of Trusted and Untrusted Code

Yi Lu
Oracle Labs Australia
November, 2015

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Introduction

- Modern programming languages are designed for internet applications and extensible systems
- Untrusted code may run in the same process as trusted code
- Fine-grained access control is needed to manage the security requirements of program code

ORACLE®

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Stack Inspection

- A permission-based mechanism for controlling code access
  - E.g., sandboxing in Java and C#

- Code attempting sensitive operations may be privileged with permissions
  - Permissions are granted to classes by policy files
  - An implementation of *the principle of least privilege*

- **All code on the call stack** must have sufficient privilege to perform certain sensitive operation
  - Tested by a permission check at runtime

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Stack Inspection Example

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      …
      l.createResource(name);
      ...
   }
}
```

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }
}
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Stack Inspection Example

```
public class A {
    public static void main(String[] args) throws Exception {
        L l = ...;
        …
        l.createResource(name);
        ...
    }
}
```

AC.checkPermission	AllPermission
L.createResource	AllPermission
A.main	ResourcePermission("*", "create")

```
public class L {
    private Resource resource;

    private native Resource create(String name);

    public void createResource(String name) {
        AccessController.checkPermission(
            new ResourcePermission(name, "create"));
        resource = create(name);
    }
}
```

ORACLE

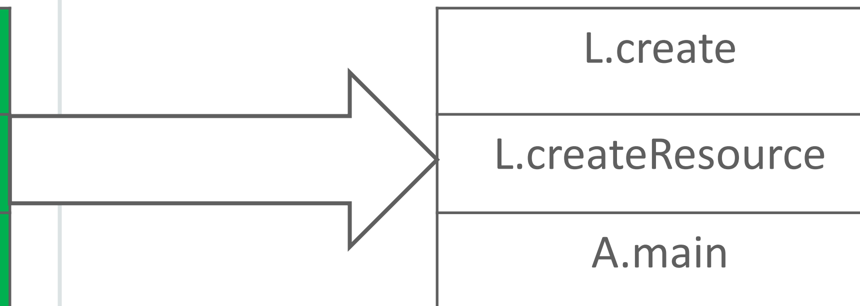
Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Stack Inspection Successful

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      …
      l.createResource(name);
      ...
   }
}
```

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }
}
```

AC.checkPermission	AllPermission
L.createResource	AllPermission
A.main	ResourcePermission("*", "create")



Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Stack Inspection Unsuccessful: Exception Thrown

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      …
      l.createResource(name);
      ...
   }
}
```

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }
}
```

AC.checkPermission	AllPermission
L.createResource	AllPermission
A.main	ϕ

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Tainted Information Used in Sensitive Operation

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...;
      String name = b.getName();
      l.createResource(name);
      ...
   }
}

public class B {
   public String getName() {
      return "password";
   }
   ...
}
```

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }
}
```

B.getName	ϕ
A.main	ResourcePermission("*", "create")

→

AC.checkPermission	AllPermission
L.createResource	AllPermission
A.main	ResourcePermission("*", "create")

ORACLE

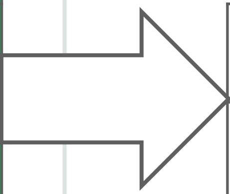
Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Leaked Sensitive Information

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...;
      ...
      Resource r = l.getResource();
      b.useResource(r);
   }
}

public class B {
   ...
   public void useResource(Resource res) { ... }
}
```

```
public class L {
   private Resource resource;

   ...

   public Resource getResource() {
      AccessController.checkPermission(
         new ResourcePermission("*", "get"));
      return resource;
   }
}
```

AC.checkPermission	AllPermission
L.getResource	AllPermission
A.main	ResourcePermission("*", "create") ResourcePermission("*", "get")



B.useResource	ϕ
A.main	ResourcePermission("*", "create") ResourcePermission("*", "get")

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Forbid Desired Operation

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      …
      l.initResource();
      ...
   }
}
```

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }

   public void initResource() {
      final String name = "initial";
      createResource(name);
   }
}
```

AC.checkPermission	AllPermission
L.createResource	AllPermission
L.initResource	AllPermission
A.main	ϕ

Security Exception

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Elevate Privilege in Java

```
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      …
      l.initResource();
      ...
   }
}
```

AC.checkPermission	AllPermission
L.createResource	AllPermission
PrivilegedAction.run	AllPermission
AC.doPrivileged	AllPermission
L.initResource	AllPermission
A.main	ϕ

```
public class L {
   private Resource resource;

   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }

   public void initResource() {
      final String name = "initial";
      AccessController.doPrivileged(
         new PrivilegedAction<Object>() {
            public Object run() {
               createResource(name);
            }
         });
   }
}
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Limitations of Stack Inspection

- Too weak against information flow attacks
  - E.g. *the confused deputy problem*
    - Untrusted code may inject data to be used by trusted code to perform sensitive operations
    - Data generated from sensitive operations by trusted code may be received by untrusted code

- Too strong to allow desired information flows
  - Often have to elevate code privilege at runtime

- Rely on programmer discipline
  - No enforceable security model or policy

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Related Work

- Stack-based access control
  - Wallach and Felten, S&P'98
  - Fournet and Gordon, POPL'02
- History-based access control
  - Abadi and Fournet, NDSS'03
- Information-based access control
  - Pistoia, Banerjee and Naumann, S&P'07

- *It is surprisingly hard to state a useful security goal that captures the intent for a general class of trusted and untrusted code*

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

A New Security Model

- Answer the questions:
  - What is the desired security goal for programs consist of trusted and untrusted code?
  - Can the security goal be enforced statically?

- Control information flow between code according to their privileges
  - Prevent undesired influences between sensitive code and unprivileged code
  - Allowing desired influences between sensitive code and privileged code

- Provide a simple and general security policy to support both confidentiality and integrity

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Connecting Information Flow to Code Access Control

- Each code is associated with a dual access control specification
  - Governing how information may flow between code

- **Capability** determines privilege/trust of code
  - i.e. the privilege granted to untrusted code

- **Accessibility** determines secrecy/sensitivity of code
  - i.e. the privilege required by sensitive code

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Example Revisited

```
@requires{} @holds{ResourcePermission("*", "create")}
public class A {
    public static void main(String[] args) throws Exception {
        L l = ...; B b = ...;
        String name = b.getName();
        l.createResource(name);
        ...
    }
}

@requires{} @holds{}
public class B {
    public String getName() {
        return "password";
    }
    ...
}
```

```
@requires{} @holds{AllPermission}
public class L {
    private Resource resource;

    @requires{ResourcePermission(name, "create")}
    private native Resource create(String name);

    public void createResource(String name) {
        AccessController.checkPermission(
            new ResourcePermission(name, "create"));
        resource = create(name);
    }
}
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Security Policy

$$x \to y \implies acc(x) \le cap(y) \wedge acc(y) \le cap(x)$$

- $x \to y$: information may flow from x to y
- $acc(x)$: the accessibility of the code defining x
- $cap(y)$: the capability of the code defining y
- A ≤ B iff B is more privileged than A

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Security Policy

$$x \to y \implies acc(x) \leq cap(y) \wedge acc(y) \leq cap(x)$$

- Privilege-dependent confidentiality
  - The receiver must have sufficient privilege to receive the information
- Privilege-dependent integrity
  - The sender must have sufficient privilege to send the information
- Mutual information flows are supported (often desired)

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Tainted Information Used in Sensitive Operation Revisited

```
@requires{} @holds{ResourcePermission("*", "create")}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...;
      String name = b.getName();
      l.createResource(name);
      ...
   }
}

@requires{} @holds{}
public class B {
   public String getName() {
      return "password";
   }
   ...
}
```

```
@requires{} @holds{AllPermission}
public class L {
   private Resource resource;

   @requires{ResourcePermission(name, "create")}
   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }
}
```

"password" $\rightarrow$ name $\Longrightarrow$ {} $\leq$ {AllPermission} $\wedge$ {ResourcePermission(name,"create")} $\leq$ {}

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Forbid Desired Operation Revisited

```
@requires{} @holds{}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...;
      ...
      l.initResource();
      ...
   }
}
```

```
@requires{} @holds{AllPermission}
public class L {
   private Resource resource;

   @requires{ResourcePermission(name, "create")}
   private native Resource create(String name);

   public void createResource(String name) {
      AccessController.checkPermission(
         new ResourcePermission(name, "create"));
      resource = create(name);
   }

   public void initResource() {
      final String name = "initial";
      createResource(name);
   }
}
```

$\text{"initial"} \rightarrow \text{name} \Longrightarrow \{\} \leq \{\text{AllPermission}\} \wedge \{\text{ResourcePermission(name,"create")}\} \leq \{\text{AllPermission}\}$

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Leaked Sensitive Information Revisited

```
@requires{} @holds{ResourcePermission("*", "get")}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...;
      ...
      Resource r = l.getResource();
      b.useResource(r);
   }
}

@requires{} @holds{}
public class B {
   ...
   public void useResource(Resource res) { ... }
}
```

```
@requires{} @holds{AllPermission}
public class L {
   @requires{ResourcePermission("*", "get")}
   private Resource resource;

   ...

   public Resource getResource() {
      AccessController.checkPermission(
         new ResourcePermission("*", "get"));
      return resource;
   }
}
```

$\text{"resource"} \rightarrow \text{res} \Longrightarrow \{\text{ResourcePermission}(\text{"}*\text{"},\text{"get"})\} \leq \{\} \wedge \{\} \leq \{\text{AllPermission}\}$

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Distinct Integral/Confidential Requirements

```
@requires{} @holds{AllPermission}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...; C c = ...;
      l.setResource(b.get());
      Resource r = l.getResource();
      c.use(r);
   }
}

@requires{} @holds{ResourcePermission("*", "set")}
public class B {
   public Resource get() { return new Resource("password"); }

}
@requires{} @holds{ResourcePermission("*", "get")}
public class C {
   public void use(Resource res) { ... }
}
```

```
@requires{} @holds{AllPermission}
public class L {
   @requires_conf{ResourcePermission("*", "get")}
   @requires_inte{ResourcePermission("*", "set")}
   private Resource resource;

   …
   public Resource getResource() {
      AccessController.checkPermission(
         new ResourcePermission("*", "get"));
      return resource;
   }
   public Resource setResource(Resource r) {
      AccessController.checkPermission(
         new ResourcePermission("*", "set"));
      resource = r;
   }
}
```

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Security Policy for Distinct Integrity/Confidentiality

$$x \to y \implies conf(x) \leq cap(y) \wedge inte(y) \leq cap(x)$$

- The receiver must satisfy the **confidential requirement** of the sender
- The sender must satisfy the **integral requirement** of the receiver

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Distinct Integral/Confidential Requirements

```
@requires{} @holds{AllPermission}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...; C c = ...;
      l.setResource(b.get());
      Resource r = l.getResource();
      c.use(r);
   }
}

@requires{} @holds{ResourcePermission("*", "set")}
public class B {
   public Resource get() { return new Resource("password"); }

}
@requires{} @holds{ResourcePermission("*", "get")}
public class C {
   public void use(Resource res) { ... }
}
```

```
@requires{} @holds{AllPermission}
public class L {
   @requires_conf{ResourcePermission("*", "get")}
   @requires_inte{ResourcePermission("*", "set")}
   private Resource resource;

   ...
   public Resource getResource() {
      AccessController.checkPermission(
         new ResourcePermission("*", "get"));
      return resource;
   }
   public Resource setResource(Resource r) {
      AccessController.checkPermission(
         new ResourcePermission("*", "set"));
      resource = r;
   }
}
```

$$\text{new Resource("password")} \rightarrow \text{resource} \Longrightarrow \{\} \leq \{\text{AllPermission}\}$$
$$\wedge \{\text{ResourcePermission("}*\text{","set")}\} \leq \{\text{ResourcePermission("}*\text{","set")}\}$$

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Distinct Integral/Confidential Requirements

```
@requires{} @holds{AllPermission}
public class A {
   public static void main(String[] args) throws Exception {
      L l = ...; B b = ...; C c = ...;
      l.setResource(b.get());
      Resource r = l.getResource();
      c.use(r);
   }
}

@requires{} @holds{ResourcePermission("*", "set")}
public class B {
   public Resource get() { return new Resource("password"); }

}
@requires{} @holds{ResourcePermission("*", "get")}
public class C {
   public void use(Resource res) { ... }
}
```

```
@requires{} @holds{AllPermission}
public class L {
   @requires_conf{ResourcePermission("*", "get")}
   @requires_inte{ResourcePermission("*", "set")}
   private Resource resource;

   ...
   public Resource getResource() {
      AccessController.checkPermission(
         new ResourcePermission("*", "get"));
      return resource;
   }
   public Resource setResource(Resource r) {
      AccessController.checkPermission(
         new ResourcePermission("*", "set"));
      resource = r;
   }
}
```

$$\text{resource} \rightarrow \text{res} \Longrightarrow \{\text{ResourcePermission}(\text{"}*\text{","get"})\} \leq \{\text{ResourcePermission}(\text{"}*\text{","get"})\}$$
$$\wedge \{\} \leq \{\text{AllPermission}\}$$

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Summary

- Stack inspection is inherently limited for secure information flow in programs consist of trusted and untrusted code
- We propose a security model to capture the intent for a general class of trusted and untrusted code
- Our security policy can be enforced by static program analysis

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Integrated Cloud

Applications & Platform Services

ORACLE

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

ORACLE®