Translating Java into LLVM IR to Detect Security Vulnerabilities

Cristina Cifuentes¹, Nathan Keynes², John Gough¹, Diane Corney¹, Lin Gao², Manuel Valdiviezo², and Andrew Gross³

¹Oracle Labs Australia, Oracle  ²Parfait, Oracle  ³Java Product Group, Oracle

What is a Vulnerability?
- Defects exploitable by malicious users
- All defects not satisfying Java Secure Coding Guidelines (JSCG) are potential vulnerabilities
- An exploit exercises one or more vulnerabilities

Caller-Sensitive Methods (CSMs)
- Permissions determined by class of immediate caller

<table>
<thead>
<tr>
<th>Stack</th>
<th>Security Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>Privileged, full resource access, must ensure unsafe objects not returned to applets</td>
</tr>
<tr>
<td>Sandboxed applets</td>
<td>Unprivileged, access limited to public resources</td>
</tr>
</tbody>
</table>

CVE-2013-0422 Security Exploit
In JDK class com.sun.jmx.mbeanserver.MBeanInstantiator:
- loadClass() uses CSM Class.forName()
- Tainted/untrusted input className used as argument
- Object theClass escapes/leaks to untrusted code

```
public Class<?> findClass(String className, ClassLoader loader)
    throws ReflectionException {
    return loadClass(className, loader);}
```

```
static Class<?> loadClass(String className, ClassLoader loader)
    throws ReflectionException {
    Class<?> theClass;
    ... try {
        if (loader == null)
            loader = MBeanInstantiator.class.getClassLoader();
        if (loader != null) {
            ...
        } else { theClass = Class.forName(className);
        }
    } catch (ClassNotFoundException e) { ... return theClass; }
```

```
@"*com.sun.jmx.mbeanserver.MBeanInstantiator.findClass(Ljava/lang/String;Ljava/lang/ClassLoader;Ljava/lang/Class;--Method" = { ... public ...}
```

Declaring that findClass is public, used by analyses to determine tainted input / information leakage (escape)

Parfait – Javac
- Uses javac to compile the source to class files
- A plugin extracts extra information from the compiler AST
- The translator produces LLVM bytecode which contains data structures to represent Java classes:
  - A structure to represent the object, i.e. instance fields
  - Global variables to represent static fields
  - Class descriptor to represent information that would be required at runtime such as: super class, methods (including visible information), fields, nested and/or enclosing classes, implemented interfaces, annotations and generic signatures
- The bytecode representation provides support for:
  - Dynamic dispatch for classes and interfaces
  - Exception handling for user-defined and runtime exceptions
  - Reflection

Parfait Analyses Based on Java Secure Coding Guidelines

Taint Analysis (Sections 9.3, 9.8, 9.9, 9.10)
- Flow and field-sensitive
- Configurable sources and sinks
- Detects reachability of untrusted data to security-sensitive operations (even via serialized fields)
- Efficient, effective, and scalable

Escape Analysis (Sections 9.8, 9.9, 9.10)
- Detects object propagation back to untrusted code
- Direct via return of public method
- Indirect via field update of a parameter

May-Null Analysis (Section 9.3, 9.9)
- Detects reference nullity
- For inferring null class loaders

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.