

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

| Stack | Security Level |
|-------------------|--|
| JDK | Privileged, full resource access, must ensure unsafe objects not returned to applets |
| Sandboxed applets | Unprivileged, access limited to public resources |

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;
}
```

This call succeeds and returns null

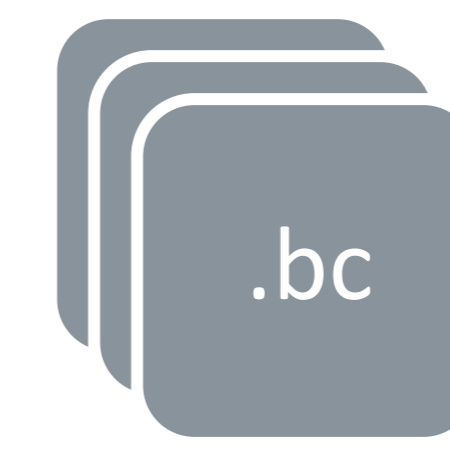
Checks class loader of immediate caller, a MBeanInstantiator instance obtained via another vulnerability

Parfait – Javac

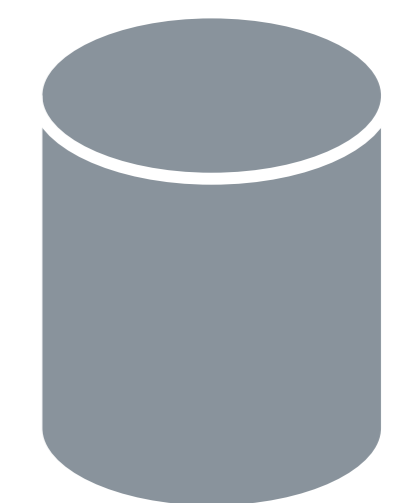
- Uses javac to compile the source to class files
- A plugin extracts extra information from the compiler AST
- The translator produces LLVM bitcode which contains data structures to represent Java classes:
 - A structure to represent the *object*, i.e. instance fields
 - *Global variables* to represent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



Java Source Code



LLVM IR



Vulnerability Reports

LLVM IR of JDK's MBeanInstantiator

Contains object-oriented metadata:

```
@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

Analyses Based on Java Secure Coding Guidelines
www.oracle.com/technetwork/java/secocodeguide-139067.html

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

className is user-defined: *tainted*

theClass *escapes* to user context

Loader is *null*: highest-privileged primordial