

“Mixed Code” security settings in Java 1.6.0_19

In march 2010 Oracle released Java 1.6.0_19 with a fix towards a security issue, reported by Signaturgruppen, that allows an attacker to piggyback malicious code with trusted applets, effectively allowing for effective drive-by attacks on the web. A security advisory has been released, with the following details:

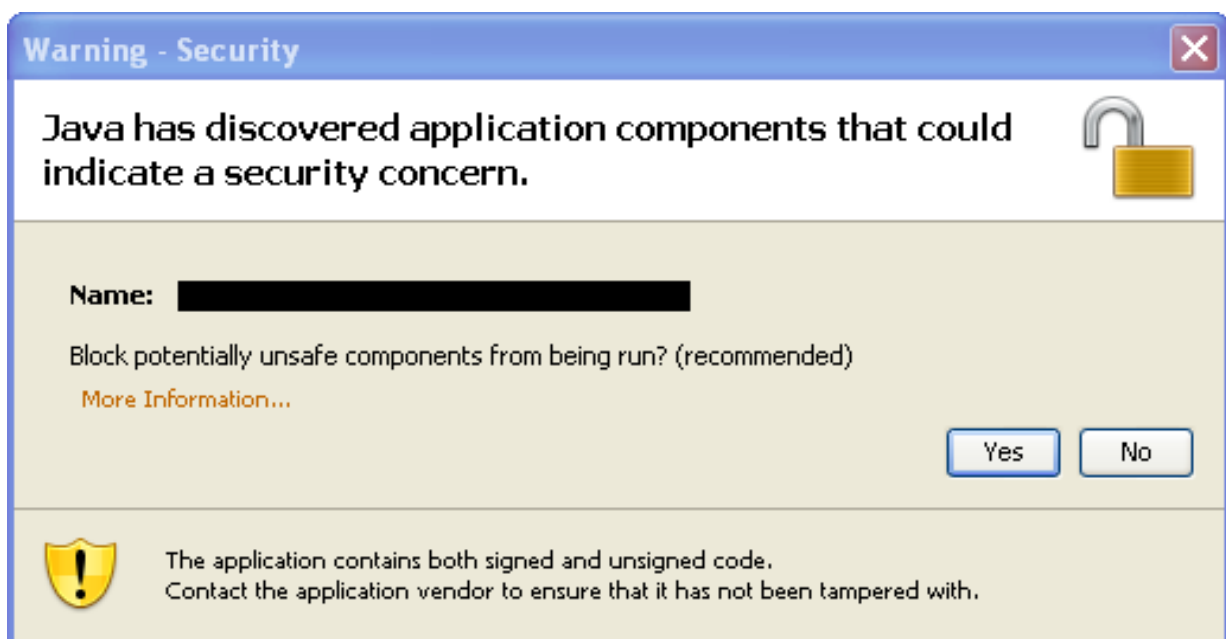
<http://secunia.com/advisories/37255/>

An error in the validation of signed Java applets can be exploited to modify the contents of a signed applet without breaking the signature. This can be exploited to execute arbitrary code outside the traditional Java sandbox if a user trusting the company signing the applet is tricked into executing it.

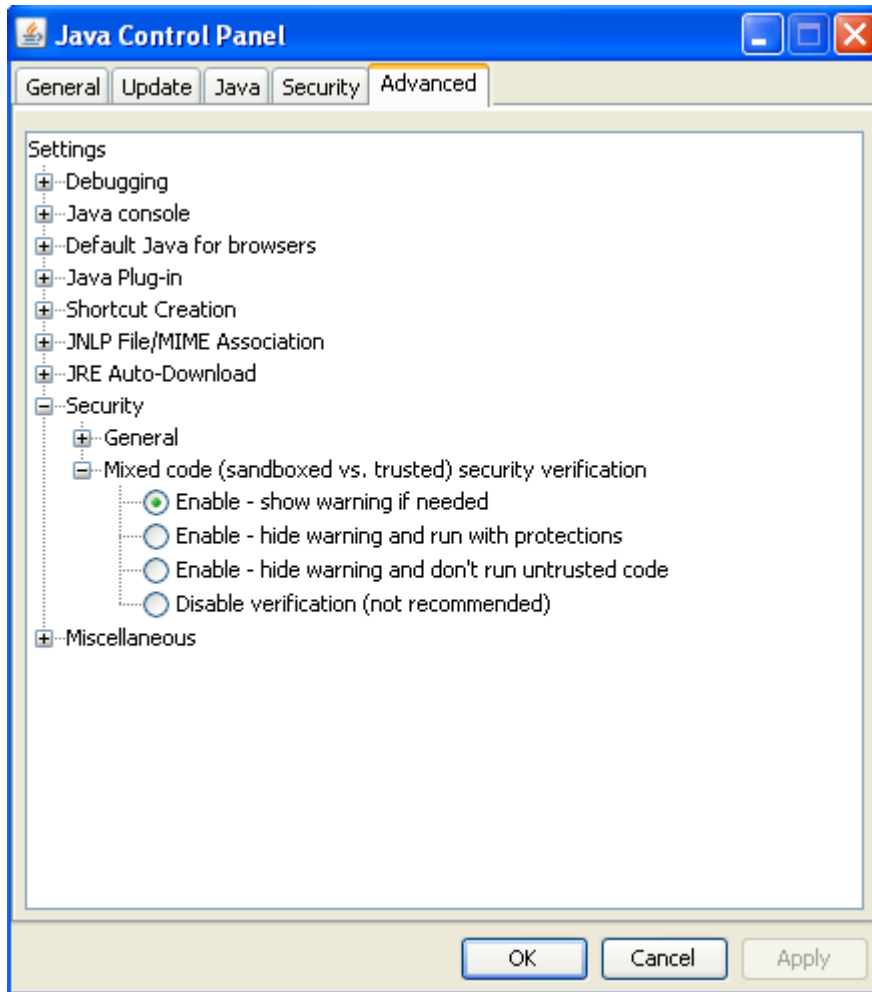
This document will highlight the effects of the security fix, and outline some of the steps that could be taken to secure applets when running with older JVMs.

The solution

To prevent the injection of malicious code, the default security settings in 1.6.0_19 will give the user a warning if signed code is executed together with any unsigned resources (code or otherwise) – the user is allowed to accept this, so legacy applications can continue to function, until the application vendor can update their application package.



Note that this is the default setting – the user may change this setting, or a company may have a different default setting. In the Java Controlpanel, the following settings are available:



The two settings in the middle will hide the warning from the user, and either (the first option) run the applet, but throw a `SecurityException` when untrusted code is executed, or simply prevent the applet from running at all (the second option). The last option will disable the entire protection scheme, leaving the user in pre- 1.6.0_19 security mode (which is not recommended for the obvious security reasons).

What does this mean for developers?

For most, this will not mean a thing. But if your applet uses unsigned resources (3rd party libraries, externally-loaded classes, external images, etc) the new security settings will likely result in a warning to your users.

Use the following list as a starting point, to ensure that your applet does not contain mixed-mode code and resources:

- All class files should be bundled in one or more JAR files, all of which should be signed, and referenced by the applet-tag like this

```
<applet archive="MyFirstJar.jar, MySecondJar.jar, etc" ....
```

- All non-code resources (property files, images, locale-files, etc) should be bundled together with the code in the signed JAR files.
- Any 3rd party libraries should either be rebundled with the existing code (and thus signed), or the library JARs should be signed.
- If a plugin architecture is used, where classes are lazy-loaded or loaded by an internal classloader, then the plugin-classes should be distributed in signed JAR packages.

How to protect users running on older JVMs

It is never a good thing to have your product associated with a security issue, and the bug in the older JVMs allows for an attacker to bundle his (unsigned) malicious code with your signed code, leveraging on the trust your customers have towards your products. The most common scenario would be the following:

1. An attacker identifies a signed applet that a lot of customers trusts, for instance an applet intended for logon to a homebanking system or similar or a nationwide digital identity scheme.
2. The attacker bundles his malicious code with the signed applet, and deploys the infected applet somewhere public, where a lot of people will see it.
3. Any visitor to any of those sites (using an older JVM) will either
 - a) Be prompted to execute an applet signed by <your company>, and if they allow the applet to run, it will execute the attack.
 - b) Have the code execute automatically, in the case where the user already trusts the signature of <your company> - this would be the common case for any customer that have previously used your applet.

The new JVM from Oracle/SUN would prevent this from happening, informing the user that the applet has been tampered with; but for users running older JVMs this is still a security issue.

The best approach would be to perform the needed signature verifications inside your own code. As part of the startup process, an applet can perform an internal verification, where it makes sure that all resources are in fact signed by the same certificate, and that no tampering on classes/resources has been performed.

An example of such an approach has been made by Signaturgruppen, where the validation process has been moved into abstract superclasses, and a secure resource loader – the changes needed to be made to an existing applet would then be minimal; effectively limited to the following



- The main (or all, if more than one is used) Applet class must extend the abstract class SecureApplet.
- All event-listeners must extend the abstract class SecureListener.
- All access to resources must be made through the class SecureResourceLoader, instead of the usual getResourceAsStream() method calls.

Or the validation routines found in these secure classes can be moved into the applets own classes.

Contact Signaturgruppen for more details on the above solution, or if you need help with updating your existing applet to conform to the security requirements of JVM 1.6.0_19.