Sand-boxing as a security measure

Pieter van den Hombergh

Fontys Hogeschool voor Techniek en Logistiek

June 6, 2016

Sandboxes in various forms.

As is Life

- A sandbox is a mechanism to restrict what a program can do.
- Typical environments have somehow to do with the way programs are distributed.
- Examples
  - Java script in a web page
  - Macro in an office document
  - javascript in pdf.
  - Java as applet in a browser plugin.
  - Flash plugin, running an app from the net.

What makes up a sandbox

- Managed execution environment.
- examples:
  - chroot environment. The app has a restricted and often private view of the "Universe" as in file system, and access to resources and IO. A modern example is docker with docker images.
  - Virtual Machine like virtual box, VMware, Virtual PC.

Leaky Sandboxes

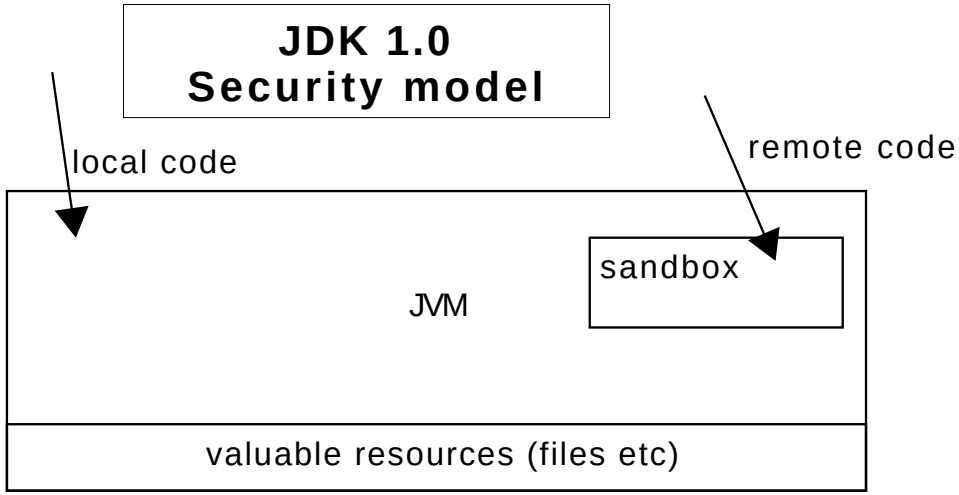
The problem with sandboxes is that they might be leaky.

- If the docker image is started with the root account, it might inherit to much authority.
- a Virtual Machine on a host is on the same (in)side of the firewall.
- The security mechanisms built into the language/sandbox may be insufficiently well designed, constructed or tested, often a combination of all three.
  - This problem is aggravated by the complexity of the interaction between virtual machine and the language and API features. Java is a good example.

Example: Java security Model

I happen to know a bit of Java

- Java has a security model since its inception. See the relevant oracle web site for that.
- It was quite Okay, because very restricted.

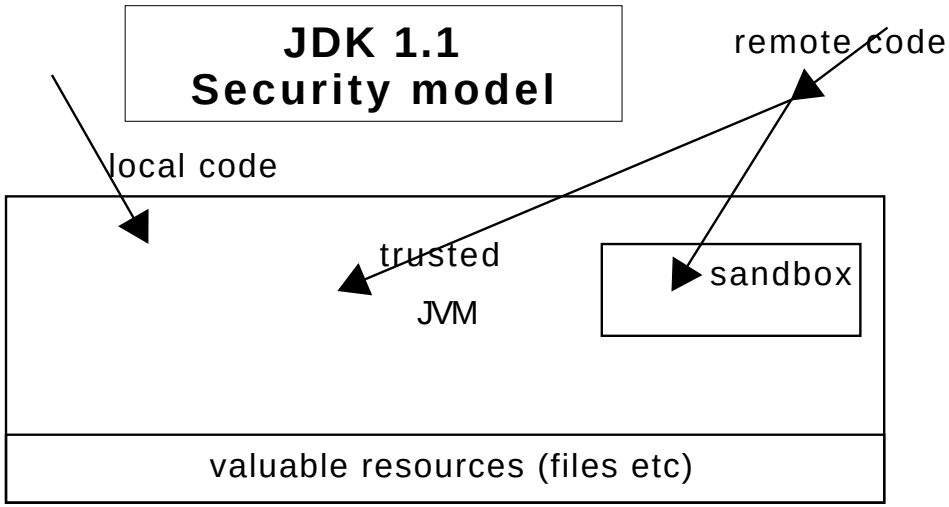


- Then other use cases arose.

Extending the use case

Can you do this too? And this...

- A app from then net needs more local access. Do we allow it?
- Enter trusted, A.K.A. signed code (with a valid certificate etc.)



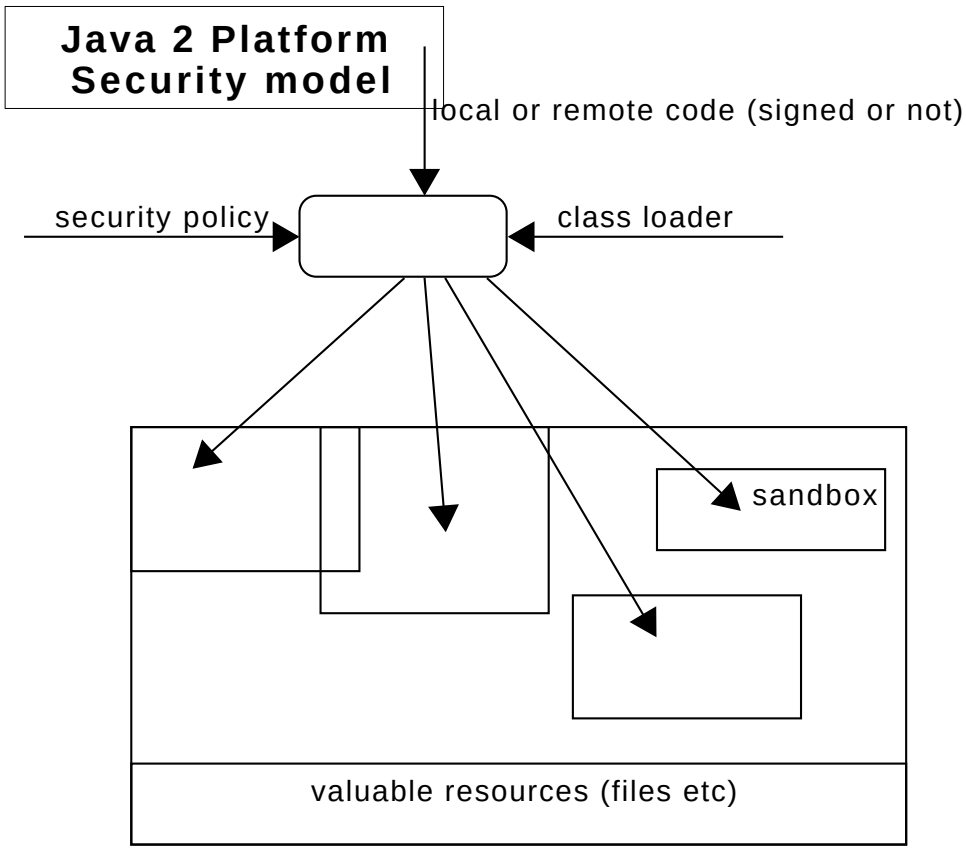
- When and why trust.

Java Security

- First of all, the language is designed to be type-safe and easy to use.
- Automatic memory management,
- Garbage collection
- Range checking on strings and arrays
- Compilers and a **bytecode verifier**
- classloader defines a **local name space** to ensure that an untrusted applet cannot interfere with the running of other programs.
- Security Manager combined with JVM can allow or disallow access to crucial system resources.

Java2 Platform security

- even more uses case, needing
- More fine grained access control (it is always about access).
- extends security policies to local code too.
- Security Manager has an API.



API Example

java.io.File#createNewFile

```
1006    public boolean createNewFile() throws IOException {
1007        SecurityManager security = System.getSecurityManager();
1008        if (security != null) security.checkWrite(path);
1009        if (isInvalid()) {
1010            throw new IOException("Invalid file path");
1011        }
1012        return fs.createFileExclusively(path);
1013    }
```

This implies that the standard applications already obeys a security manager, iff enabled.

By the way, the package private (the default) visibility is in fact stronger guarded and enforced by the JVM then the "ordinary private and protected" visibility. The fact that each class loader can bring its own name space makes this model only stronger.

Problems and Consequences

- Java 7 introduced some goodies, as well as some less so.
- In particular the Java Reflection API was revised and extended.
  - With some serious issues, costing Oracle some considerable time to fix and delayed the arrival of Java 8.
  - Java is now no longer considered as a viable platform to distribute applications in the form of Applets, once one of its fortes.
  - Best you can do in that direction is signed web-start applications.
- Desktop and server applications do NOT have the problems that need-t-sandbox applications have. They are installed and hence trusted by the administrator or run in a designated environment (special daemon user).