SecurityManager介绍

Java序列化漏洞利用中突出的一点是，一旦一个服务器端的Java应用程序被破解，那么下一步就是获取主机上的shell访问权限，这就是我们所熟知的远程代码执行（RCE）。

然而有趣的是，其实从Java 1.1开始，Java中就存在一种方式来限制代码执行，并能够防止远程代码执行，这就是SecurityManager。在启用SecurityManager之后，Java代码会在一个非常安全的沙盒中执行，而这能够防止远程代码执行。

```
java  -Djava.security.manager  com.example.Hello
```

这会以$JAVA_HOME/jre/lib/security/java.policy中的默认安全策略运行，在JDK 1.8中为如下内容：

```
// Standard extensions get all permissions by default

grant codeBase "file:${/*}" {
        permission java.security.AllPermission;
        };
// default permissions granted to all domains

grant {
               // Allows any thread to stop itself using the java.lang.Thread.s
top()
               // method that takes no argument.
               // Note that this permission is granted by default only to rem
ain
               // backwards compatible.
               // It is strongly recommended that you either remove this permi
ssion
               // from this policy file or further restrict it to code source
```

```
s
                // that you specify, because Thread.stop() is potentially unsaf
e.
                // See the API specification of java.lang.Thread.stop() for mor
e
                // information.
                permission java.lang.RuntimePermission "stopThread";

                // allows anyone to listen on dynamic ports
                permission java.net.SocketPermission "localhost:0", "liste
n";

                // "standard" properies that can be read by anyon
e

                permission java.util.PropertyPermission "java.version", "rea
d";
                permission java.util.PropertyPermission "java.vendor", "rea
d";
                permission java.util.PropertyPermission "java.vendor.url", "rea
d";
                permission java.util.PropertyPermission "java.class.version", "rea
d";
                permission java.util.PropertyPermission "os.name", "rea
d";
                permission java.util.PropertyPermission "os.version", "rea
d";
                permission java.util.PropertyPermission "os.arch", "rea
d";
                permission java.util.PropertyPermission "file.separator", "rea
d";
                permission java.util.PropertyPermission "path.separator", "rea
d";
                permission java.util.PropertyPermission "line.separator", "rea

d";
                permission java.util.PropertyPermission "java.specification.versio
n", "read";
```

```
                permission  java.util.PropertyPermission  "java.specification.vendo
r",  "read";
                permission  java.util.PropertyPermission  "java.specification.name",  "re
ad";
                permission  java.util.PropertyPermission  "java.vm.specification.versio
n",  "read";
                permission  java.util.PropertyPermission  "java.vm.specification.vendo
r",  "read";
                permission  java.util.PropertyPermission  "java.vm.specification.nam
e",  "read";
                permission  java.util.PropertyPermission  "java.vm.version",  "rea
d";
                permission  java.util.PropertyPermission  "java.vm.vendor",  "rea
d";
                permission  java.util.PropertyPermission  "java.vm.name",  "read";
};
```

以下面的[代码]举例：

```
package  com.example
object  Hello  {
        def  main(args:  Array[String]):  Unit  =  {
                val  runtime  =  Runtime.getRuntime
                val  cwd  =  System.getProperty("user.dir")
                val  process  =  runtime.exec(s"$cwd/testscript.sh")
                println("Process  executed  without  security  manager  interferenc
e!")
          }
}
```

策略文件

在启用安全管理器并使用一个额外的[策略文件]时，可以明确地启用或禁用执行特权：

```
grant  {
        //  You  can  read  user.dir
        permission  java.util.PropertyPermission  "user.dir",  "read";
        //  Gets  access  to  the  current  user  directory  script
        permission  java.io.FilePermission  "${user.dir}/testscript.sh",  "execute";
```

```
        permission  java.util.PropertyPermission  "scala.control.noTraceSuppression",  "r
ead";
};
```

你可以以下面的方式运行：

```
java  -Djava.security.manager  -Djava.security.policy=security.policy  com.example.Hello
```

如果只注释掉FilePermission那一行，那么将会抛出一个异常。

到目前为止一切都比较顺利。但是，那只是对客户端一侧的applet启用了这个功能，此时在服务器端仍然是禁用状态。

这是为什么呢？其实，因为（可以查看上述代码）默认的SecurityManager将系统锁在了无用点上。为了使系统处于有用状态，它必须拥有一个自定义的java.security.policy文件。

问题分析

这种策略的实现中有几个问题。因为策略文件本身是过时了的，安全权限并未以任何一种逻辑顺序进行排序，且一些权限拥有通配符的选择功能而其他的则没有。你可以以白名单的形式只允许特定的行为，而不是直接拒绝。最坏的情况是，列表越长，那么程序将运行得越慢。这里有一个教程和权限列表，但是在实践中并不是特别有用，并且这份文档指南上一次的更新时间早在2002年。

当你有了不可信任的代码时，可以编写定制的SecurityManager，而这就是Scalatron所做的工作。

然而，如果我们要防止远程代码执行，那么我们需要一个通用目的的SecurityManager，其中它几乎允许所有事情，但是能够阻止在主机上运行的代码。这可能不是一个完美的防守，但是它将是深度防御策略中很重要的一部分。

事实上，已经有个人做了这一点。

pro-grade介绍

Josef Cacek 将pro-grade集成在一起，向Java文件中添加了一个“deny”选项和一个“allow”选项。Devoxx中有一个演讲以幻灯片共视频形式讲解了pro-grade。

现在，使用pro-grade进行前面的示例，下面的策略将会锁定所有执行权限：

```
priority  "grant";
```

```
deny {
        //https://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html#FilePermission permission java.io.FilePermission "<<ALL FILES>>", "execute";
};
```

注意，这不是一个完整的解决方案。为了阻止代码工作，我猜想你还需要禁用其他几个权限，不过我还不清楚哪些权限是与此相关的。但是，这只是一个开始，为了廉价地确保服务器端的Java程序安全，还有很长的路要走。

使用Pro-grade很简单创建一个适当的策略。有一个策略生成器可以显示一个应用程序所需要的所有权限，且一个教程展示了创建它所需要的所有步骤，然后有一个权限调试器能够捕捉偏离的权限。

应用到你的项目中

你可以在prograde-example这里得到整个项目，并且可以从这里将pro-grade集成到你的项目中。此外，大多数人将需要Maven：

```
<dependency>
        <groupId>net.sourceforge.pro-grade</groupId>
        <artifactId>pro-grade</artifactId>
        <version>1.1.1</version>
</dependency>
```

pro-grade真正有趣的地方是，它是一个透明的解决方案。虽然使用白名单策略是一种不错的方式，但是使用这种技术你可以将pro-grade添加到一个现有的、已经编译的项目目中，并且可以禁用脚本执行。

只需一些小小的修改，pro-grade就可以用来通知入侵（尤其是setSecurityManager和其他很少接触的地方），似乎在默默地矫正操作时也能够正常工作。你需要做的所有工作就是以一个SLF4J实现来实现PermissionDeniedListener。

***参考来源：tersesystems，FB小编JackFree编译，转载请注明来自FreeBuf黑客与极客（FreeBuf.com）**