Quiz yourself: Using the SecurityManager class in Java

JAVA SE

Quiz yourself: Using the SecurityManager class in Java

Be sure to use the checkPermission and doPrivileged methods correctly.

by Mikalai Zaikin and Simon Roberts

March 8, 2021

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

Imagine you are creating an application that reads sensitive information from remote hosts for use by trusted parts of the application. To reduce network overhead, that sensitive information is stored in a cache; requests are fulfilled directly from the cache whenever possible. You will use Java's security system to control access to this sensitive data.

Given this method:

```
public static String getSecret(String host) {
    Permission perm = getHostPermission(host)

    if (distributedCache.containsKey(host)) {
        return distributedCache.get(host);
    }

    AccessController.checkPermission(perm);

    PermissionCollection perms = perm.newPerm
    perms.add(perm);

    PrivilegedAction<String> pa = new Privile
        public String run() {
            return getSecretFromHost(host);
```

```
        }
    };

    AccessControlContext acc = new AccessCont
        new ProtectionDomain[] {
            new ProtectionDomain(null, perms)
        }
    );

    String secret = AccessController.doPrivil

    distributedCache.put(host, secret);
    return secret;
}
```

Which step below will best protect the application? Choose one.

A. Move line n2 up to be placed right after line n1.

The answer is A.

B. Remove line n2.

The answer is B.

C. Replace line n3 with
`String secret = AccessController.doPrivileged(pa, null);`
.

The answer is C.

D. Replace line n3 with
`String secret = AccessController.doPrivileged(pa);`
.

The answer is D.

Answer. Java has had a strong and well-designed security infrastructure from its very beginning, although the use of those features in mainstream applications sadly remains relatively uncommon.

A Java class called `SecurityManager` can be installed in a JVM. When this is done, that security manager will check for any potentially security-sensitive operations—and prevent such operations if they are not explicitly permitted.

Many kinds of operations can be security-sensitive, such as opening a file for reading or changing the security configuration of the system. Another example would be opening a display window, since that might be used to create a fake login screen.

The mechanism of these security checks is both very fine-grained and extensible. It is *fine-grained* in that permission might

be granted for one specific file to be read but not written to. *Extensible* means that programmers can define new permissions and integrate the security manager's checks into their own code.

A key part of the system is the notion of *permissions*. Every piece of code loaded into a JVM has a set of permissions associated with it. Any one thread might pass through many different pieces of code as it makes method calls. When a thread attempts to execute any potentially sensitive operation, an access check determines whether the necessary permission is available and, if it's not, the attempt is aborted with an exception.

In a simple case, a permission is available if every method on the thread's call stack has that permission. The checks are embedded in Java's libraries, and they act as gatekeepers for all such operations. The location of the checks is very close to the actions themselves and carefully designed to ensure the checks cannot be bypassed.

The set of permissions associated with a particular piece of code is configured in a policy file and can differ based on several things: the origin of the code, any recognized signatures carried by the JAR file from which the code was loaded, and who is running the code. (That "who" might be an identity; it also might be a machine rather than a human.)

Using these mechanisms, it's possible to give a different set of permissions to third-party libraries than those given to in-house code, and a different set again if code has been dynamically loaded from a remote location, as was the case with the now mostly forgotten applet concept.

Using permissions, it is possible to list the operations that a piece of code is *intended* to perform and allow it to perform only those operations. Any other operation that it might attempt, either because of a programming error or because of malicious actions, will then be rejected. This type of explicit permission-granting is far safer than the normal behavior of allowing the program to do whatever it wants, restricted only by the user identity running the code. A wise saying in security is "that which is not explicitly permitted is denied."

As already mentioned, the permission checks that are performed typically test the permissions associated with the entire call stack. This is important! Imagine that an untrusted piece of code calls a trusted piece of code in such a way that the trusted code might read a file. If this can happen, there is a chance that the untrusted code might simply ask the trusted code to do something malicious on its behalf. As a result, the general behavior is that when an attempt is made to, for example, open a file, all the methods on that thread's call stack must have that permission. If even one method does not, the call is rejected.

However, sometimes it's necessary for a trusted piece of code to be able to do just this kind of behavior on behalf of untrusted code. Imagine untrusted code trying to present a piece of text on

a display. It needs a font to render the pixels of the message, and fonts are read from files. In this situation, there is a mechanism that allows the trusted code to say, in effect, “I’m helping out here, and I have determined that what I’m doing cannot be abused, so let me use my trusted privilege to do this, even though my caller, which doesn’t have that trust, wouldn’t be able to do it without my help.”

The basic checking mechanism outlined above is embedded in a method called `checkPermission`. The mechanism that allows trusted code to exercise its permissions when acting on behalf of less-trusted code is embedded in a method called `doPrivileged`.

Okay; now let’s get back to the quiz question, which has a method named `getSecret` that gives access to a sensitive item of data. That method has the use of a cache and also of the remote source of the original data. The goal is to ensure that only properly trusted code can obtain the secret data. This must involve a permission check that occurs regardless of whether the secret is returned directly from the cache or after being read from the network.

As it stands, code that lacks privileges will be able to obtain secrets that are already in the cache. This is an error; you should ensure that the secret itself, not merely the network transaction that fetches it, is protected. In view of this, the proper place for the `checkPermission` call is at the top of the `getSecret` method, and this means that option A offers a significant security improvement.

Option B, which removes the permission check from line n2, would make matters much worse. The code as presented strongly suggests that accessing the secret should have the permission `perm` that is calculated on line n1. Certainly, the question states that the goal is to prevent untrusted parts of the application from accessing this secret. Now, if line n2 were removed, callers without the permission would be able to borrow the permission by the `doPrivileged` block, which would allow the network operation to succeed and the untrusted code would gain access to the secret. From this you conclude that option B is severely incorrect.

The proposals of options C and D are equivalent to one another; that is, they both modify the behavior of the `doPrivileged` call in the same way. Both of these calls lend the entire permission set associated with the `getSecret` method to the execution of the `PrivilegedAction` that does the network access. That, of course, is essentially the point of the `doPrivileged` method, so it might at first seem that these changes would make no difference. However, the existing call in the sample code actually causes the `doPrivileged` block to run with *only* the `perm` permission.

By contrast, the code suggested by options C and D will invoke the network access with *all* the permissions associated with the `getSecret` method. That's likely to be far more privileges than necessary, and the principle of least privilege suggests that's a bad choice. It might not change anything in the code as it exists, but it fails to close doors that have no reason to be open. So, both options C and D actually reduce the overall security of the system and, therefore, options C and D are incorrect.

At this point, you know that option A not only improves matters but is the only option that does. Therefore, option A is correct.

For further reading, check out chapter 9, "Access control," in the *Oracle secure coding guidelines for Java SE*.

Conclusion: The correct answer is option A.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Share this Page