Code Access Security

by Kip Irvine

Last update: 4/21/2015
Contents

1. Code Access Security concepts
2. Code Access Security in Java
3. .NET
4. Java Security Policy Files

© Kip Irvine, 2003-2012. All rights reserved.
Security Issues

- Runtime code may originate from many sources in a networked environment.
  - email, documents, web apps
- Users can be authenticated by logon credentials, and therefore trusted, but they may inadvertently execute corrupted or malicious code.
- A security mechanism is needed that allows code originating in one system to run on another system, even when there is no trust relationship.
Code Access Security

- Allows code to be trusted to varying degrees, depending on
  - where the code originates
  - if it is digitally signed
  - type of requested access to target system
- Minimizes the amount of code that must be fully trusted.
- Almost always relies on a virtual machine
How Does it Work in .NET?

- When a managed code program executes, the target code is evaluated by CLR first, and assigned a permission level.
  - either allowed to run, or a security exception is generated
  - local computer settings ultimately decide which permissions to give to the program
How to Write Secure Applications

- Use a compiler that generates type-safe code (Java, C#, VB). *Do not use C or C++.*
- Call the runtime security system functions, using one of these:
  - declarative, using attributes
  - imperative, using instantiated objects
- Request permissions for your code
  - evaluated at runtime by the class loader
- Use secure class libraries

Requests cannot influence the runtime to give your code more permissions than the runtime would have given your code had the request not been made. But you use the request to inform the runtime about the permissions you require in order to run.
Type-Safe Code

- References to object fields must be at fixed offsets, corresponding to named field members
  - avoid offsets that can be modified at runtime
- Languages that generate native code cannot be verified as type-safe
Declarative Security Syntax (.NET)

- Use attributes to place security information into your code's metadata.
  - may appear at the assembly, class, or member level

- Example:

```csharp
[MMyPermission(SecurityAction.Demand, Unrestricted = true)]
public class MyClass {
    public MyClass() {
        //The constructor is protected by the security call.
    }

    public void MyMethod() {
        //This method is protected by the security call.
    }
}
```
Imperative Security Syntax (.NET)

- Create a new instance of the permission object you want to invoke.
  - initialize the permission object with information about how it will be used

- Example: the System.Security.FileIOPermission class
  - Demand(): Forces a SecurityException at run time if all callers higher in the call stack have not been granted the permission specified by the current instance.
Imperative Security Syntax (.NET)

Example:

```csharp
public class MyClass {
    public MyClass() {
    }
    public void MyMethod() {
        // MyPermission is demanded using imperative syntax.
        MyPermission perm = new FileIOPermission("document.txt");
        perm.Demand();
        // This method is protected by the security call.
    }
    public void YourMethod() {
        // This method is not protected by the security call.
    }
}
```
Code Access Security in Java
Java Security

- Java Sandbox security model (3 levels)
  - bytecode verifier
  - class loader
  - security manager
Bytecode Verification

- After class loader executes, class passed to the Java virtual machine (VM)
- bytecodes are inspected by a verifier that checks for buggy or damaging code.
- Can be disabled:
  
  ```
  java -noverify classname
  ```
Bytecode Verification

- **Examples:**
  - variables not initialized before being used
  - method calls match types of object references
  - rules for accessing private data and methods followed
  - local variable accesses are inside the runtime stack
  - stack overflow

- **Doesn't the compiler already check these?**
  - class file could be altered
  - see: VerifierTest.java (from Core Java 2 book)
VerifierTest

Java source code:

```java
static int fun()
{
    int m;
    int n;
    m = 1;
    n = 2;
    int r = m + n;
    return r;
}
```
VerifierTest Example

- Run the *javap* utility to disassemble:

```
javap -c VerifierTest
```

```
Method int fun()
  0 iconst_1
  1 istore_0 ; m = 1
  2 iconst_2
  3 istore_1 ; ** will be modified **
  4 iload_0
  5 iload_1
  6 iadd
  7 istore_2
  8 iload_2
  9 ireturn
```
VerifierTest Example

- Compile the program
- Modify the bytecode
  - Change `istore_1 (0x3C)` to `istore_0 (0x3B)`
- Run the program:

```java
java VerifierTest
Exception in thread "main" java.lang.VerifyError: (class: VerifierTest, method: fun signature: ()I) Accessing value from uninitialized register 1
```
Security Policy Files

- developer grants permission for a program to execute certain operations
  - grants varying levels of access to system resources
  - default level is very restrictive
- Permissions granted on basis of:
  - code signer (using signedBy)
  - source of the code (using codeBase)
- You can use the policytool utility
Filename: *authorized.policy*. The statement here grants permission for a program to write to a file named *authorized.txt*.

```java
grant {
    permission java.io.FilePermission
        "authorized.txt", "write";
};
```
public class MyFileWriter {
    public static void main( String args[] ) {

        System.setSecurityManager( new SecurityManager() );

        FileWriter writer = new FileWriter( "authorized.txt" );
        String someText = "..."
        writer.write( someText );
        writer.close();
    }
}
Executing MyFileWriter

Throws a java.security.AccessControlException:

```java
MyFileWriter
```

Exception in thread "main" java.security.AccessControlException: access denied
(java.io.FilePermission authorized.txt write)

```
at java.security.AccessControlContext.checkPermission(Unknown Source)
at java.security.AccessController.checkPermission(Unknown Source)
at java.lang.SecurityManager.checkPermission(Unknown Source)
at java.lang.SecurityManager.checkWrite(Unknown Source)
at java.io.FileOutputStream.<init>(Unknown Source)
at java.io FileOutputStream.<init>(Unknown Source)
at java.io.FileWriter.<init>(Unknown Source)
at MyFileWriter.main(MyFileWriter.java:15)
```

Executes correctly:

```bash
java -Djava.security.policy=authorized.policy MyFileWriter
```
Readings

- Code Access Security (article) at