

Securing Java Application Servers

Kirill Bolshakov
Leonid Mokrushin

{raven,leom}@dcn.nord.nw.ru
Distributed Computing & Networking Dept.
Saint-Petersburg State Technical University



Abstract

We present a set of concepts to enable application service providers (ASP) to host untrusted Java/XML/XSP-based applications. A set of patches for JServ engine is presented that includes custom security manager, as well as a standalone kind of this security manager. It enables server administrator to use flexible access control mechanism to finely tune access to various parts of the server. It also enables secure hosting of multiple applications without mutual trust relationships inside one servlet engine. The concepts are demonstrated on the example of Cocoon-based multi-user XSP application server.

Good morning! Let me introduce myself. My name is Kirill Bolshakov, and I am currently with Distributed Computing and Networking Department of Saint-Petersburg State Technical University. In this talk, I will present a set of concepts as well as a project on security of Cocoon/XSP-based application hosting. The project, named "Xemis" (this word is a hybrid between "XML" and "Semis"), enables application service providers to host multiple XSP applications originating from different publishers, without code reviews and suchlike.



Session Agenda - 1

- Application hosting
 - CGI
 - Java Servlets
 - Cocoon/XSP
 - Security issues
- Java 2 Platform Security Model
- Xemis
 - Project goals
 - Security model
- Building and running Xemis

Kirill Bolshakov, Leonid Mokrushin

2

Let me describe session agenda. First, we will discuss several application hosting schemes, namely, CGI scripts, Java servlets and Cocoon technology. We will look at them in order to highlight security and maintenance issues. After that I will talk about Java 2 Platform security model to remind you of the latest version of Java sandbox.

Next, I will proceed to the project itself. I will declare project goals and present its security model. The brief installation guide will be the next topic.

Session Agenda - 2

- Security policies notation
- Example: accessing database
 - Allow everything!
 - Add restrictions
 - Debugging the policy set
- Benchmarking Xemis
- Project status
- Further development

Kirill Bolshakov, Leonid Mokrushin

3

To enforce something you have to express it first, so then I will proceed to the notation of security policies that are used to configure Xemis. Then I will comment on some simple example and demonstrate techniques that should be used to configure Xemis properly. Description of benchmarking results, project status and planned development will go afterwards.

We Are Now At...

- Application hosting
 - CGI
 - Java Servlets
 - Cocoon/XSP
 - Security issues
- Java 2 Platform security model
- Xemis
 - Project goals
 - Security model

Now we will proceed to discussing application hosting techniques, their pros and cons.



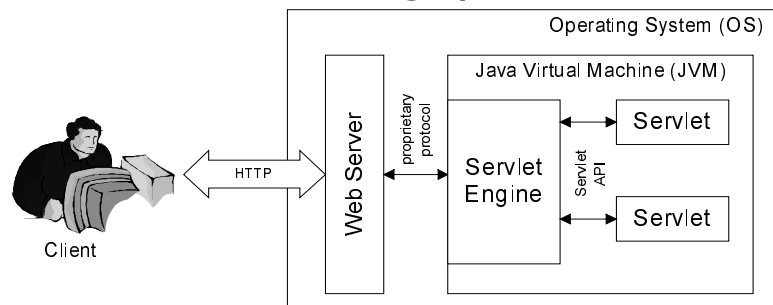
Application Hosting: CGI Scripts

- Functionality is limited by OS API only
- Rely on native OS security mechanisms
- CGI scripts are OS-dependent
 - Application server becomes OS-dependent
 - Significant amount of maintenance
- Full-featured programs
 - Development takes much time

The first hosting method I am going to discuss is CGI scripts. The scripts are not necessarily written in scripting language, and may be complex programs in C or C++. Their functionality is limited only by imagination of the developer and OS API. The security mechanisms in use are those of the OS. This way, highly integrated environments emerge, like those of Microsoft Windows and Microsoft IIS Web server. However, the drawbacks here are the following: the application server has to be OS-dependent, and the amount of maintenance in case of heterogeneous hosting is rather large. Also, remember all those buffer overflows with stack smashing? The developer has to be very careful while writing even the simplest C++ Web bulletin board. Thus, the time of the development is increased.

Application Hosting: Java

- Platform-independence and faster development
- EJB as enterprise-oriented standard
- Application can be secured, but what about non-dedicated hosting system?




Kirill Bolshakov, Leonid Mokrushin

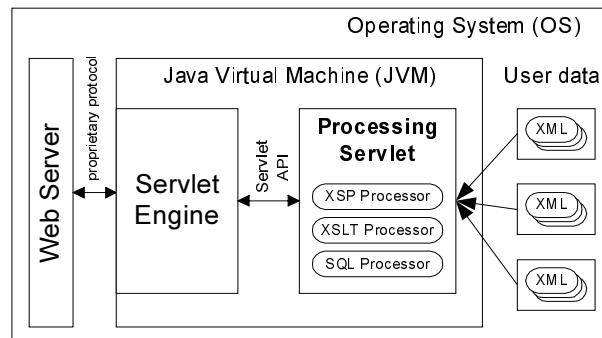
6

In the case of Java servlets, we see major advance in their platform-independence (this fact virtually turns any heterogeneous network into homogeneous from point of view of the servlet). Java is also much more “fault-tolerant” than traditional compiled languages: the absence of pointers and array bounds checking imply the absence of buffer overflows and stack smashing. Automatic garbage collection speeds up the development. Being an object-oriented language, Java forces to use OOA/OOD paradigms, and this is good as well. EJB as enterprise-oriented standard help to build protected applications without thinking of their security in its whole — there is a way to restrict access to beans’ methods.

Let’s consider request’s life in such a case. First, the request travels to the Web server. After being mapped to servlet engine, it produces a thread within servlet engine. The engine loads necessary servlet and invokes its, for example, `doPost()` method. Consider a number of servlets executing within the same JVM. They can interfere with each other; access each other’s files and secret scripts (e.g., those with database passwords). Such kind of a situation helps us to formulate the question: is it possible to protect Java application hosting system in the case of multiple publishers?

Application Hosting: Cocoon/XSP

- Java-based  Platform-independent
- Separates content, style and logic
- Easily extendable
- Trivial publishing
- Almost zero maintenance



Kirill Bolshakov, Leonid Mokrushin

7

Cocoon/XSP hosting solution looks very attractive. First, Cocoon paradigm explicitly separates content, style and logic of the application. Second, it is Java-based, and all Java constructs may be used in XSP programs, for the programs are compiled to Java. And third, the maintenance is near zero: "Setup and forget". However, all the flaws in security area are inherited from Java servlets.

The extensibility of Cocoon framework allows easy creation of custom producers and processors. XML acts like a glue between different parts of the application.

XSP stands for eXtensible Server Pages. This technology allows the developer to put his application in an XML document. The application will have access to the structure of the document as if it were written with knowledge of existence of this document. However, the application resides INSIDE the document, and Cocoon performs all code generation and compilation tasks itself. This feature makes XSP one of the most powerful and easy to maintain Web programming technologies.

Security Issues

- CGI scripts
 - Native OS security
 - Platform-dependent maintenance
 - Java servlets
 - What about multiple publishers?
 - Cocoon/XSP
 - Same as Java: unable to securely host applications from different publishers inside one JVM
-

Kirill Bolshakov, Leonid Mokrushin

8

Let me summarize security and maintenance features of the above-mentioned hosting solutions. CGI scripts utilize native OS security mechanisms, and their maintenance is cumbersome. They do not work in heterogeneous environments. Java servlets allow creation of safe network applications, but they lack security mechanisms in the case of multiple publishers. Cocoon/XSP inherits this drawback from Java servlets (for it is still a servlet) but wins in maintenance area. Should we try to solve the problem, we should definitely look for the solution for Cocoon/XSP, because this very technology offers power combined with maintenance simplicity.

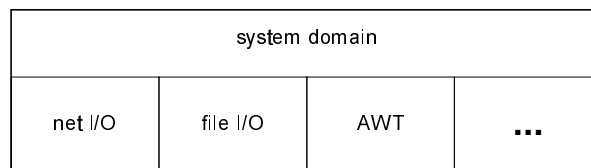
We Are Now At...

- Application hosting
 - CGI
 - Java Servlets
 - Cocoon/XSP
 - Security issues
- Java 2 Platform security model
- Xemis
 - Project goals
 - Security model

Now we will describe JDK 1.3 Security Model. Then, we will proceed to Xemis project: its goals and its security model.

Java 2 Platform Security Model

- “Sandbox” model
- Different codes run with different permissions
- “Package-centered” security
- Based on protection domains: sets of classes whose instances are granted the same set of permissions

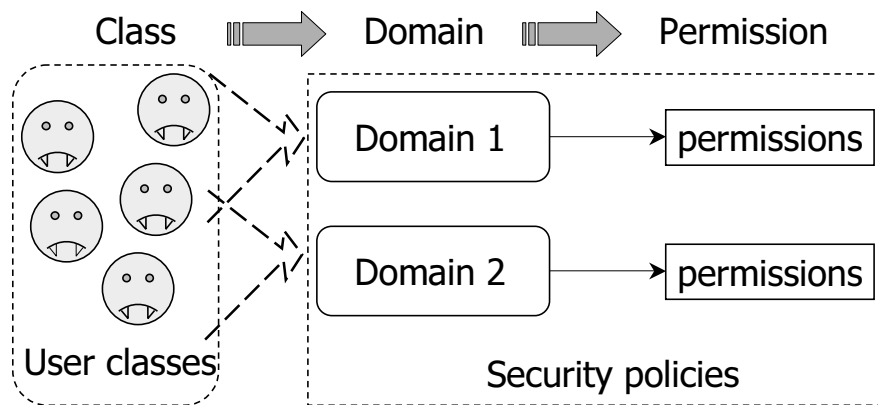


Kirill Bolshakov, Leonid Mokrushin

10

Here I will remind of Java 2 security model. At the base of Java 2 security model lays the notion of “Sandbox”. Sandbox serves as a layer between valuable system resources (such as file system) and application programs. Java application cannot access the resources directly. Thus, this layer can perform security checks: if the instance of the class that is trying to access the resource should be allowed to do that. The code is assigned to domains basing, for example, on the package it is in, or the physical location it resides at. The resources are split into domains as well — you can see an example of such domains in the above picture. Here, applications can access network input/output, file system and AWT only through using system domain. At the layer of system domain security checks take place. All Java libraries which work with resources of native OS or with JVM internals use this paradigm to protect the system from malicious applications.

Java 2 Platform Security Model - 2



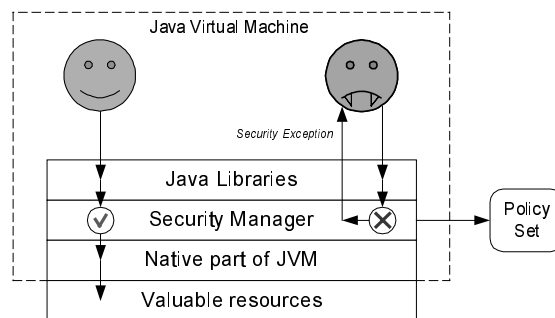
Kirill Bolshakov, Leonid Mokrushin

11

A domain conceptually encloses a set of classes whose instances are granted the same set of permissions. Protection domains are determined by the policy currently in effect. The Java application environment maintains a mapping from code (classes and instances) to their protection domains and then to their permissions, as illustrated by the figure above. The permission system is initialized basing on the Java security policies given in configuration files. This gives flexible way for setting up security for Java applications.

Java 2 Platform Security Model - 3

- Security Manager
 - Extendable via inheritance



Kirill Bolshakov, Leonid Mokrushin

12

The security check is performed in the following way: the thread that is making an attempt to access the resource is led to `SecurityManager` object. `SecurityManager` has a set of methods corresponding to various actions that can be applied to various resources. This object checks thread's privileges (for example, basing on the stack contents of this thread and determining the "least powerful class instance" in the stack) and either grants the permission or throws `java.lang.SecurityException`. A thread that can access the resource just continues its execution and goes to the native part of JVM.

For example, if some thread is trying to perform `READ` operation on a file or file handle, `SecurityManager.checkRead()` method will be invoked by system library code responsible for reading from files with its argument set to name of the file being accessed. Then, after looking at the call stack, `SecurityManager` decides whether or not this thread will be given access to the file.

In some cases, `SecurityManager` has to traverse all the stack looking for class instance belonging to domain with least privileges. If all class instances in the stack have been checked and there were no "trespassers", `SecurityManager` simply performs "return" from this method. Otherwise, `java.lang.SecurityException` is thrown.

Java 2 Platform Security Model - 4

- Security Manager
 - Extendable via inheritance
- Policy-based configuration

```
grant codeBase "file:/c:/ApacheJServ/-" {
    permission java.lang.RuntimePermission "createSecurityManager";
    permission java.lang.RuntimePermission "setSecurityManager";
    permission java.lang.RuntimePermission "modifyThread";
    permission java.net.SocketPermission "127.0.0.1:1024-", "accept,
    resolve";
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.util.PropertyPermission "*", "read";
};
```

Kirill Bolshakov, Leonid Mokrushin

13

One interesting feature of Java 2 Security architecture is that the standard security manager can be overridden in order to allow extension of security mechanisms. This means that it can be extended in the way of introducing one's own security paradigm. However, the resources and actions stay the same in the case when the security manager has been overridden.

The rules that grant or revoke access are given in the form of policy files (introduced by Sun). For example, in the picture above a permission to create ClassLoader and modify threads (i.e., thread's name) is granted to instances of classes under c:/ApacheJServ folder. Instances of these classes can also:

- Create and set SecurityManager in the current instance of JVM (by the way, SecurityManager is a JVM-wide object).
- Accept and listen to connections with the peers that connect to the localhost.
- Read all system properties.

codeBase is the place from which classes being checked originate.

Protected Objects

- AWT
- File
 - Access to a file or directory
- Network
 - Socket operations
 - URL handling
- Security
 - Policy manipulation, etc.
- Property
 - JVM properties
- Reflect
 - For reflective operations
- Runtime
 - ClassLoader
 - SecurityManager
 - JVM, etc.

Kirill Bolshakov, Leonid Mokrushin

14

The above picture lists those JVM and OS objects access to which is regulated by Java security policies (the list is incomplete). Access to these objects is specified using corresponding permissions. AWT permissions include access to the clipboard, display pixels, and AWT event queue. File permission controls read/write/execute and delete access to files and folders. Network permission controls the flow of authentication information during URL fetches; socket permission includes control of accept(), connect(), listen() and resolve() operations. Security permission controls manipulation of system-wide access policies and management of cryptography service providers. Property permission controls access to JVM properties. Runtime permission controls all run-time related actions, such as creating the ClassLoader, halting the JVM, etc.

Here we come to the conclusion that the set of protected objects is fine-grained.

Publisher-oriented Security

- Desired protection scheme
 - System objects must be protected from hosted applications
 - Hosted applications must be protected from each other
 - Individual access rights should be granted on per-publisher basis
 - Keep policy-based management intact

To solve the problem posed earlier we need to establish some degree of understanding of what features should be present in the solution. It is clear that the first two goals: “System objects must be protected from hosted applications” and “Hosted applications must be protected from each other” implies the third one. The way must be found to identify the publisher of the executing XSP program and then to apply corresponding set of policies. Additionally, it is desirable to preserve currently existing method for expressing permissions.

Xemis: Project Goals

To provide:

- Complete Java application hosting solution
- Fine-grained security for multiple publisher environments
- Administrator-friendly configuration tools
- Developer-friendly deployment and security debugging tools

Xemis project is aimed not only at providing such fine-grained security for multiple publisher environments. The tools for application deployment should be developed as well. Also, the developer should have his application prepared to run in restricted environments. For instance, he should be able to easily learn all the resources that are accessed by his application during run-time. Should he be aware of the list of accessed resources he may compare it with the server policy and introduce corresponding corrections to his application or request policy change from server's administrator. The developer should also be able to easily diagnose problems occurring with his application after it has been deployed to the server. The administrator should have convenient graphical tool for specification and validation of policy sets.

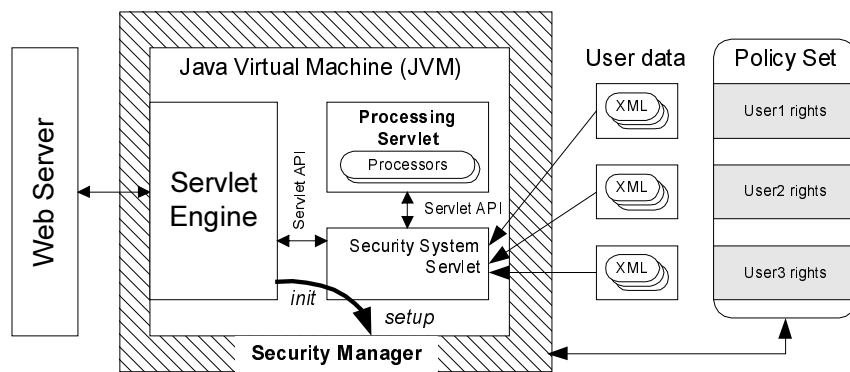
Xemis: Security Model

- Publisher's name is extracted from requested URL
- Request identity is tracked throughout request lifetime
- Access permissions for most objects can be specified w.r.t. publishers

Given the above conditions, one of the solutions looks like the following: XSP programs should be placed in directories corresponding to their publishers; the name of the publisher should be extracted from request URL and then used to identify request thread. Runtime permissions should be setup in the way that this thread should be unable to change its name. This approach will allow identifying publisher of the XSP program that was invoked first: further attempts to get round the security will result in `SecurityException` because of inability of the thread to change its name. In the next slide, we show the place of Xemis in Cocoon/JServ blend.

Xemis Implantation

- Patch for Servlet Engine (Apache JServ)
 - Parses the URL and sets thread name
- Overridden SecurityManager



Kirill Bolshakov, Leonid Mokrushin

18

Xemis should be installed as the servlet that starts immediately after Servlet Engine. This allows installation of custom SecurityManager before any other servlets will load. The Servlet Engine has been modified to recognize requests that address publishers' "home" directories. This being the case, due to knowledge about server directories layout, JServ is able to extract publisher's name and assigns request thread's name to it. After this action, there is no way to change the name of the request's thread. During Security Manager methods invocations the name of the publisher can be obtained from the name of the thread.

To provide debugging capabilities and enhanced error reporting, XSP programs should be fed to Cocoon by Xemis. This is why we introduced new handler to Apache configuration: XML files with XSP programs should now have ".xms" extension.

Security Policies Notation

- Keywords add-ons
 - Files-related
 - \$ALL_FILES\$
 - Publisher-related
 - \$EVERYBODY\$
 - \$NOT_A_USER\$
 - Action-related
 - action_read
 - action_write
 - action_execute
 - action_delete
 - action_all
- Policy files add-ons
 - fileAccess.policy
 - classAccess.policy
 - packageAccess.policy
 - packageDefine.policy

Kirill Bolshakov, Leonid Mokrushin

19

Xemis introduces several security policy files. `classAccess.policy` contains rules on which classes are allowed to access which classes. `packageAccess.policy` contains rules on which classes are allowed to access which packages. `packageDefine.policy` contains rules on which classes may define which packages. This file is usually empty for the task of defining some package is usually performed by ClassLoaders in some special cases.

The general rule to remember is: what is not explicitly granted is implicitly denied. Try to describe all possible situations when you are going to grant more access for some application publisher.

The most interesting and useful is `fileAccess.policy`. It defines publishers' access to file system. There are a number of variables that may be used within this file. An example of `fileAccess.policy` file is demonstrated on the following example.



Security Policies Notation - 2

■ fileAccess.policy fragment

```
# xemis administrator permissions
C:\Apache\htdocs\users\admin\ - = admin, action_all
C:\Xemis\conf\* = admin, action_all

# xemis users permissions
C:\ApacheJServ\servlets\zone.properties = $EVERYBODY$, action_read
C:\Apache\htdocs\users\ - = $EVERYBODY$, action_read
C:\Apache\htdocs\users\${USER_NAME}\private\ - = raven, leonidm,
  action_read

# standalone servlets permissions
C:\ApacheJServ\servlets = $EVERYBODY$, action_read
C:\ApacheJServ\servlets\ - = $EVERYBODY$, action_read
```

Consider this fragment of fileAccess.policy file.

First section of the file grants full access to Xemis configuration and admin's files to administrative user. In the second section, the first line grants read-only access to zone.properties to all users. The second line grants access to the root of users home directory. The third line contains special variable \$USER_NAME\$, which, during security checks, is substituted with user name from the right side and then compared to the real situation: who wants to get access to whose private files. Third section of the file grants everybody access to "well-known" servlets that do not belong to any publisher.

We Are Now At...

- Security policies notation
- Example: accessing database
 - Allow everything!
 - Add restrictions
 - Debugging the policy set
- Benchmarking Xemis
- Project status
- Further development

Now we will study a small example of database access. However, I would call it “terse” – the amount of work it performs is tremendous.

Building and Running Xemis


- Available at <http://xemis.sourceforge.net>
- Prerequisites:
 - Apache Web Server
 - Sun JDK 1.3
 - Apache JServ / Sun JSDK
 - Cocoon
 - Optional: MySQL

Xemis depends on significant amount of software packages. However, it directly relies on JServ only, for it needs patched version of JServ for it to function properly. Patched version of JServ is important for Xemis only. If you ever decide to remove Xemis from your system, you can leave patched version of JServ in place.

The following packages should be obtained:

- Apache Web Server (1.3.12)
- Sun JDK 1.3
- Sun JSDK
- Cocoon (<http://xml.apache.org>)
- Apache JServ (please look into the README file in Xemis distribution — it may contain important updates on Xemis—JServ compatibility)
- As an optional component, you can download and install MySQL (however, you can use ODBC-JDBC bridge for studying the example)

Building and Running Xemis - 2

- Install Apache Web Server
- Install and test JServ
- Apply thread-naming patch to JServ
- Install and test Cocoon
- Install Xemis as a servlet
- Tune JServ/Apache parameters
- Modify Xemis access policies
- Watch for `java.lang.SecurityExceptions!` 

Kirill Bolshakov, Leonid Mokrushin

23

This slide roughly describes Xemis installation procedure once you have all packages downloaded. We strongly recommend perform installation procedure step-by-step with testing every step. First, install Apache Web Server. Install JServ and test whether your installation was successful. Then apply thread-naming patch to JServ, rebuild it and test again. Then follow on-slide instructions and instructions in Xemis install.txt file.



Example: Accessing DB

```
<?xml version="1.0"?>
<?cocoon-process type="sql"?>
<?xml-stylesheet href="sql.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<?cocoon-format type="text/html"?>
<connectiondefs>
  <connection name="schedule">
    <driver>org.gjt.mm.mysql.Driver</driver>
    <dburl>jdbc:mysql://10.0.101.85/SPbSTU</dburl>
    <username>teacher</username>
    <password>THEWALL</password>
  </connection>
</connectiondefs>
<query connection="schedule">
  select * from tutors
</query>
</page>
```

- XSP page
- Uses SQL and XSL processors
 - Connects to external database
 - Accesses XSL in the file system
- Uses HTML formatter

Now we will consider an example of simple database access. However, the simplicity is deceptive:

- `<?cocoon-process type="sql">` invokes SQL processor which fetches data from external database `"jdbc:mysql://10.0.101.85/SPbSTU"`;
- `<?xml-stylesheet href="sql.xsl" type="text/xsl"?>` tells XSLT processor about stylesheet, which should be applied to the result of SQL lookup;
- `<?cocoon-process type="xslt">` invokes XSL Transformation processor;
- `<?cocoon-format type="text/html"?>` invokes Cocoon HTML formatter and displays the page.

Thus, it is very useful to produce a set of policies that will allow this application to perform its task but will not allow doing anything else.

In the next slides we will consider several boundary situations ("mostly allowed" and "mostly denied") in policy files.

Allow Everything!

- classAccess.policy

```
- = -
```

- fileAccess.policy

```
$ALL_FILES$ = $EVERYBODY$, action_all
```

- java.policy

```
grant { permission java.security.AllPermission; };
```

- packageAccess.policy

```
- = -
```

This example is equivalent to the absence of SecurityManager. In fileAccess.policy everybody is given full access to all files. In classAccess.policy and packageAccess.policy all classes are allowed to access any other classes and packages. In java.policy all the permissions, which are controlled by standard SecurityManager, are granted to all classes.

Add Restrictions

■ classAccess.policy

```
# allowing Xemis to access any classes
org.xjtek.xemis.- = -

# allowing system classes to access other system classes and Cocoon repository
sun.-          = *, java.-, sun.-, org.apache.-, _C_.Apache._htdocs._users.-
java.-         = *, java.-, sun.-, org.apache.-, _C_.Apache._htdocs._users.-
org.apache.-   = *, java.-, sun.-, org.apache.-, _C_.Apache._htdocs._users.-
javax.servlet.- = *, java.-, sun.-, org.apache.-, _C_.Apache._htdocs._users.-
org.gjt.mm.mysql.- = *, sun.io.-, java.text.-

# allowing user classes to access system classes
_C_.Apache._htdocs._users.- = java.-, sun.-, org.apache.-, _C_.Apache._htdocs._users.-
```

■ packageAccess.policy

■ packageDefine.policy

Kirill Bolshakov, Leonid Mokrushin

26

This example is the one where most “non-standard” (i.e., DB lookup, network connection) actions will be considered a security violation. Note the second section. If it is absent, JServ and Cocoon are unable to function properly (for example, Cocoon will be unable to invoke Java compiler). Java compiler resides inside sun.tools package. In this case, org.apache.- (Cocoon) is given access to sun.- (Javac), and sun.- (Javac) is given access to _C_.Apache._htdocs._users.- (user compiled XSP scripts).

Add Restrictions - 2

■ java.policy

```
grant codeBase "file:${java.home}/lib/ext/*" { permission java.security.AllPermission; };
grant codeBase "file:/c:/jdk1.3/lib/*" { permission java.util.PropertyPermission "*",
    "read"; };
grant codeBase "file:/c:/Xemis/-" { permission java.security.AllPermission; };
grant codeBase "file:/c:/ApacheJServ/-" {
    permission java.lang.RuntimePermission "createSecurityManager";
    permission java.lang.RuntimePermission "setSecurityManager";
    permission java.lang.RuntimePermission "modifyThread";
    permission java.net.SocketPermission "127.0.0.1:1024-", "accept, resolve";
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.util.PropertyPermission "*", "read";
};
...
grant codeBase "file:/c:/Cocoon/bin/-" {
    permission java.lang.RuntimePermission "createClassLoader";
    permission java.util.PropertyPermission "*", "read";
};
...
```

Kirill Bolshakov, Leonid Mokrushin

27

This example explicitly lists all the permissions that are required for normal operation of *system* utilities, such as JDK, Xemis (granted all permissions), JServ (granted permissions to connect to Apache, read system properties, use ClassLoader, setup Security Manager and label threads) and Cocoon (granted permission to use ClassLoader). This fragment does not contain any notion of user codebases. This implies that user classes are extremely restricted in their actions.

Add Restrictions - 3

■ fileAccess.policy

```
# xemis administrator permissions
C:\Apache\htdocs\users\admin\ - = admin, action_all
C:\Xemis\conf\* = admin, action_all

# xemis users permissions
C:\ApacheJServ\servlets\zone.properties = $EVERYBODY$, action_read
C:\Apache\htdocs\users\ - = $EVERYBODY$, action_read
C:\Apache\htdocs\users\$USER_NAME$\private\ - = raven, leonidm, action_read

# cocoon repository permissions
C:\Apache\repository\C\_Apache\htdocs\_users\_USER_NAME$ = raven, leonidm,
  action_read, action_write
C:\Apache\repository\C\_Apache\htdocs\_users\_USER_NAME$\ - = raven, leonidm,
  action_read, action_write
C:\Apache\repository\C\_Apache\htdocs\_users = $EVERYBODY$, action_read, action_write
...
```

Kirill Bolshakov, Leonid Mokrushin

28

This file grants access to file system. Please note the way Xemis users permissions are granted. Users are even unable to write to their own repositories. When dealing with Cocoon repository, however, we have to grant write permission, because otherwise all compiler invocations will fail because of the inability to write output files to the file system.

Because the layout of Cocoon's repository directly depends on layout of Web server file tree, we were able to write rules for access of individual users to their and only their part of Cocoon repository. Otherwise, some malicious code could overwrite classes of the user with higher privileges and then gain unauthorized access to important information.

Debugging the Policy Set

- Error messages for XMS scripts

```
User = leonidm

java.lang.SecurityException: Writing to the file c:\test.txt has been denied in debug mode. Use Xemis debugger.
    at org.xjtek.xemis.util.DebugSecurityManager.checkWrite(DebugSecurityManager.java:444)
    at java.io.File.createNewFile(File.java:689)
    at C:\_Apache\htdocs\_users_\leonidm\_scripts\_test2.tryToWriteFile(_test2.java:46)
    at C:\_Apache\htdocs\_users_\leonidm\_scripts\_test2.populateDocument(_test2.java:215)
    at org.apache.cocoon.processor.xsp.XSPPage.getDocument(XSPPage.java:96)
    at org.apache.cocoon.processor.xsp.XSPProcessor.process(XSPProcessor.java:456)
    at org.apache.cocoon.Engine.handle(Engine.java:305)
    at org.xjtek.xemis.Xemis.service(Xemis.java:201)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:588)
    at org.apache.jserv.JServConnection.processRequest(JServConnection.java:317)
    at org.apache.jserv.JServConnection.run(JServConnection.java:188)
    at java.lang.Thread.run(Thread.java:484)
```

Kirill Bolshakov, Leonid Mokrushin

29

The first thing you will get when trying to tune security policies for your site and your applications will be a `SecurityException`. Security exceptions are caught by Xemis (as it serves as a handler for .xms files) and are displayed to the user.

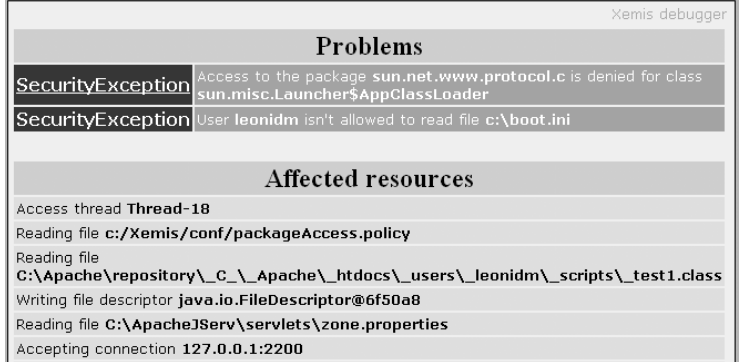
In this example file write operation for `c:\boot.ini` was denied while being in Debug Mode.

Do not worry that you do not understand why your program has taken this or that action and, as a result, tried to access some forbidden resource. Go to the Admin screen and click “Install Debug Security Manager”. Next slide explains what Debug Security Manager is.

Debugging the Policy Set - 2

- Debug SecurityManager: allows almost anything while emitting warnings...

```
File c:\boot.ini:
[boot loader]
timeout=2
default=multi(0) disk(0) rdisk(0) partition(1) \WINNT
[operating systems]
multi(0) disk(0) rdisk(0) partition(1) \WINNT="Windows NT Workstation Version 4.00"
multi(0) disk(0) rdisk(0) partition(1) \WINNT="Windows NT Workstation Version 4.00 [VG]
```



Xemis debugger

Problems	
SecurityException	Access to the package <code>sun.net.www.protocol.c</code> is denied for class <code>sun.misc.Launcher\$AppClassLoader</code>
SecurityException	User <code>leonidm</code> isn't allowed to read file <code>c:\boot.ini</code>

Affected resources

```
Access thread Thread-18
Reading file c:/Xemis/conf/packageAccess.policy
Reading file
C:\Apache\repository\C_\_Apache\htdocs\_users\_leonidm\_scripts\_test1.class
Writing file descriptor java.io.FileDescriptor@6f50a8
Reading file C:\ApacheJServ\servlets\zone.properties
Accepting connection 127.0.0.1:2200
```

Kirill Bolshakov, Leonid Mokrushin

30

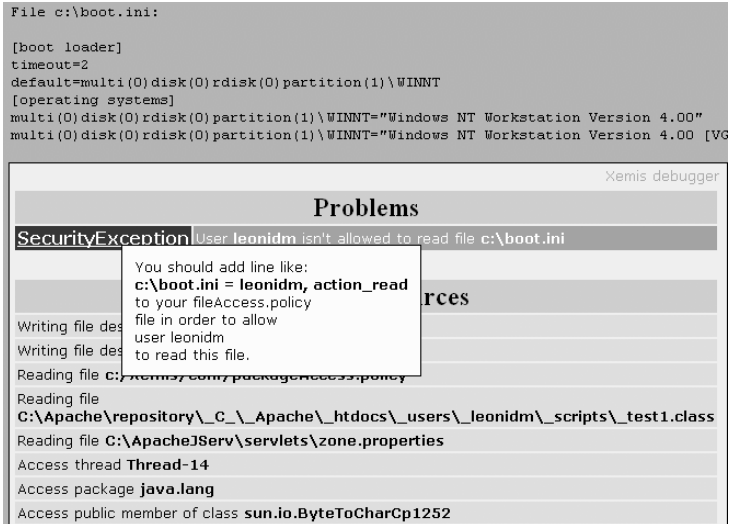
Xemis Debug Security Manager is the utility you should use not only when you get `SecurityException`s, but also for taking a closer look at what resources your application is trying to use. It allows almost all operations while emitting a warning on every current policy set violation and a note on every resource access attempt. Using it, you can easily catch unexpected `SecurityException`. For example, your application can try to load some class while you do not know about it: this is a usual situation with Java. Also look carefully through “Affected Resources” section, for it may reveal *potential* problems.

In this example, there were two security exceptions. The first one took place because of restriction to package access. One system class was unable to access another system class. The second one happened because read permission of `c:\boot.ini` was not granted to anyone.

Debugging the Policy Set - 3

- And providing recommendation:

```
File c:\boot.ini:
[boot loader]
timeout=2
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00"
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00 [VG
```



The screenshot shows the Xemis debugger interface. At the top, the file `c:\boot.ini` content is displayed. Below it, a 'Problems' window shows a `SecurityException` with the message: 'User leonidm isn't allowed to read file c:\boot.ini'. A tooltip provides a recommendation: 'You should add line like: `c:\boot.ini = leonidm, action_read` to your fileAccess.policy file in order to allow user leonidm to read this file.' Below the exception, a list of operations is shown, including 'Writing file des...', 'Reading file C:\xemis7.com\packageAccess.policy', 'Reading file C:\Apache\repository\C_Apache_htdocs_users_leonidm_scripts_test1.class', 'Reading file C:\ApacheJServ\servlets\zone.properties', 'Access thread Thread-14', 'Access package java.lang', and 'Access public member of class sun.io.ByteToCharCp1252'.

Kirill Bolshakov, Leonid Mokrushin

31

You can also click on the exception to get a suggestion on how to avoid it. The Debug Security Manager will suggest you the line you should add to one or another policy set to enable this operation. In this case, leonidm was denied access to `c:\boot.ini`. Debug Security Manager suggested that you should add "`c:\boot.ini = leonidm, action_read`" line to your `fileAccess.policy` file to avoid this exception.

Currently, these tool tips work in Microsoft Internet Explorer only.

Debugging the Policy Set - 4

- Defaults are not allowed for critical operations:
 - OS Command execution
 - Writing to files
 - Deleting files
- Command-line utility is provided
 - “Default answer: Yes” mode supported

During debugging the following displeasing accident may occur: because of the logical mistake the program will invoke “`rm -rf /`”, or delete “`/etc/passwd`”, or write garbage to the same location. To avoid these accidents, Debug Security Manager does not allow these operations until you have console debug client running (start it by running `Xemis.jar`). It shows question about whether or not should it permit OS command execution, writing to file or deleting the file. If you are sure that nothing horrible will happen to your system during debugging of the program, you can switch the client to “Default answer: Yes” mode.

We Are Now At...

- Security policies notation
- Example: accessing database
 - Allow everything!
 - Add restrictions
 - Debugging the policy set
- Benchmarking Xemis
- Project status
- Further development

In the final part of the session the benchmarking results will be presented. Current project status as well as possible further developments will be outlined.

Benchmarking Xemis

- Configuration:
 - PPro200x2, 96M RAM, Windows NT 4
- Consumes 14ms vs. 1ms on `fileAccess` check operation
- Other checks
 - Contain less string comparisons
 - Execute faster

We have benchmarked current version of Xemis and obtained the following times on the most computation-intensive test. Compared to those of Cocoon page processing, we consider these to be convenient for totally non-optimized piece of software. ☺ Other checks have less computations so the difference is not that meaningful. There is a vast field for optimizations in this area (b-trees, fast strings comparison, policy caching, etc.)



Availability. Current State

- Available at <http://xemis.sourceforge.net>
- Implemented components:
 - Security Manager
 - Debug Security Manager
 - Benchmarking sample

Kirill Bolshakov, Leonid Mokrushin

35

Stable packaged versions, as well as development version (with CVS access) and bug database are accessible at <http://xemis.sourceforge.net>.

The following components have been implemented:

1. Xemis Security Manager
2. Debug Security Manager
3. Benchmarking sample (let us know your results, please)

Using the first two components one can build secure Cocoon/XSP multi-user hosting site.

Further Development

- Optimization of security checks
- XSP deployment tools
- GUI tool for policy specification
- Research of consistency checking for policy sets

At the moment, the following directions of further development and enhancement of Xemis are under consideration:

1. Optimization of security checks. There is a huge amount of string comparisons in certain parts of the SecurityManager, so this part can be optimized in speed at the expense of memory.
2. Development of secure deployment tools for Cocoon/XSP applications.
3. Development of graphical tool for specifying policy sets. This can help an administrator to faster setup and easier manage his installation of Cocoon/Xemis blend.
4. Research on consistency checking for policy sets. The results of this research may prove to be useful for finding human mistakes when configuring the system.

Acknowledgements

- Distributed Computing and Networking Department of Saint-Petersburg State Technical University
- Experimental Object Technologies (www.xjtek.com)
- Apache Software Foundation
- Sun Microsystems
- SourceForge.NET

Thank you!

We would like to thank Distributed Computing and Networking Department of Saint-Petersburg State Technical University and Experimental Object Technologies group for the opportunity they gave us to develop this piece of software. We are also grateful to Sun Microsystems for the change they have introduced to the world of heterogeneous computing and Apache Software Foundation for their great free software. Our thanks go to SourceForge for we are hosting our project at SourceForge.NET, and to all of you for your attention.