DisCo Runtime: Secure Code Distribution and Dynamic Runtime Permissions in a P2P Environment

Hesham W. Wahba, Eric Freudenthal

July 6, 2003

1 Abstract

The need to distribute code securely while authenticating the receiver and still allowing for redistribution and appropriate “sandboxing” of code running on a host to prevent malicious or simply inappropriate access is growing. This project aims to allow for such capabilities by developing a simple abstraction and a seamless implementation in the Java language that will enable applications to take advantage of these capabilities without any re-compilation on their end. By taking advantage of the components of the DisCo project that can establish secure, authenticated connections [1] and provide continuous permission monitoring [2], as well as standard Java subsystems such as the serialization portion of RMI, we have been able to accomplish this goal, and work is ongoing to refine this model further.

2 Introduction

Various secure communication and RPC communication schemes have been proposed for P2P systems, however the issue of secure code distribution has remained largely unexplored. How one is to ensure that code gets to those who need it in a secure and seamless manner is the goal of this project, as well as developing a means of providing a sandbox to protect the consumers of this code from any malicious or unauthorized actions taken by it.

DisCo Runtime is a self-contained Java module that one need only reference in the scripts that run an application, and it will seamlessly provide the user with a secure means of downloading and running code. Since this project takes place in the context of the DisCo project, middleware for security aware applications that operate in a heterogeneous networked environment, we leverage as many components in this project as possible without giving up on the generality of the code used. We use Discovery to find the nearby authorized publishers of the code we are looking for, Switchboard [1] to distribute the code securely, and
dRBAC to dynamically monitor the permissions of the code which will now be running in a sandbox. Application developers are of course free to plug in any method they like to distribute, authenticate, and monitor code based on their own security constraints.

3 Abstraction

For the purposes of the DisCo Runtime module, Secure Code Servers (SCS) distribute code to authorized hosts. A SCS serves conventional Java class files, but is also aware of: the discover tags associated with such code so that others can try to locate it on a network, the authorization requirements for hosts receiving the class files and any of the credentials they need to complete the code download and establish a secure execution context with appropriate permissions. The SCS distributes code securely by sending encrypted class files, or by sending files over an encrypted connection such as a Switchboard connection which allows for mutually authenticated secure streams. This ensures that the code distributed doesn’t get into the wrong hands by listening in on an authorized consumer’s connection. A SCS can be established on any host with the correct authorization.

Codebase discovery keys contain enough information to both identify and validate authorized SCS, and can be used in the future in the establishment of a SCS on a host as well. Using a discovery mechanism, an execution environment can search for and get any code that the code it is running needs (using something like the DisCo locality-aware discovery system, for example) by discovering an authorized SCS and determining whether or not code from that SCS is authorized to run on the host. The execution environment also knows how to establish the proper execution parameters such as local file system or network access. These rights are monitored for the lifetime of the object.

When an object is passed between hosts after being serialized appropriately, using MarshalledObject in Java, the full codebase including the SCS discovery tags are transparently communicated to the receiving end so that code distributed in the future will be distributed properly and allowing for appropriate execution permissions to be established. See Figure 1 for more details.

4 Implementation

The Java implementation is broken up into a number of interconnected components:

- ContainerClassLoaderSpi: A wrapper that allows us to use our ClassLoader for MarshalledObjects in Java
- ContainerClassLoader: A ClassLoader that can understand our special codebases, download code securely, and setup a secure Container for the code to run in
Implementación

- **ContainerPermissionsCollection**: Una extensión de la colección de permisos predeterminada de Java `PermissionCollection` que nos permite modificar los permisos de las clases cargadas dinámicamente.

- **PermissionsAdaptor**: Traduce los permisos del `AuthorizationMonitor` para su uso por la `ContainerPermissionsCollection` para permitir permisos dinámicos de código ejecutable.

Por medio de la utilización de las hook de serialización y carga de clases en RMI, hemos sido capaces de utilizar estos módulos sin tener que reacomodar el cliente; simplemente establecemos el RMIClassLoaderSpi como el `ContainerClassLoaderSpi`. El `ContainerClassLoaderSpi` es simplemente un contenedor alrededor de nuestro `ContainerClassLoader` que hace la mayoría de la trabajo pesado. Vea la Figura 2 para más detalles.

Cada vez que un objeto es serializado, un método de serialización en Java, el objeto se anota con un código base definido en el lado del servidor. Este código base es un tipo especial de URL que se ve como una dirección HTTP para que Java pueda satisfacerla, pero en realidad contiene un descriptor o localizador que puede ser utilizado por el `ContainerClassLoader` para encontrar el código. Cuando el cliente recibe el objeto, este será procesado por el `ContainerRMIClassLoaderSpi` en el que el `ContainerClassLoader` es invocado. El `ContainerClassLoader` luego intenta encontrar un servidor de código que ofrezca el código. Si el servidor de código se considera aceptable, el código será seguramente descargado y un `Container` es creado para manejar los permisos dinámicos del código descargado. El `Container` es simplemente una abstracción que usamos en la implementación de Java, y realmente su funcionalidad es ganada al unir un `PermissionsAdaptor` al código que traduce los permisos desde un `AuthorizationMonitor` en el cliente. El `AuthorizationMonitor` es corrido por el cliente, y contiene los permisos que el cliente desea que tenga el código ejecutable. El magia de permisos dinámicos es proporcionada por la `ContainerPermissionsCollection` que nos permite modulaciones permisos de una clase en el momento en estos traducidos por el `PermissionsAdaptor` del `AuthorizationMonitor`. 
5 Summary

This project is still in development, but what we have so far allows for secure distribution of code, and modification of the runtime permissions of that code based on the host and the code server, and it is all seamlessly integrated with previously written applications. This code can be used currently to support sandboxed applications on kiosks for example, or to allow content-providers to ensure that only valid users of their data can access it. Users can remain secure in the knowledge that whatever does run on their systems will not cause any damage.

References
