Information Society Technologies

S3MS Security of Software and Services for Mobile Systems

3.2.2- Inlining Tool

Irem Aktug (KTH)

Document Number	3.2.2
Document Title	Inlining Tool
Version	2.2.0
Status	Final
Work Package	WP 3.2
Deliverable Type	Prototype
Contractual Date of Delivery	28/02/2008
Actual Date of Delivery	13/03/2008
Responsible Unit	KTH
Contributors	KTH
Keyword List	Run-time Monitoring
Dissemination level	PU

Change History

Version	Date	Status	Author (Unit)	Description
1.0.0	21/9/2007	First Draft of M18	Irem Aktug (KTH)	
2.0.0	10/3/2008	Working Copy of M24	Irem Aktug (KTH)	Updated to include new features such as .jar file input and changes in the command that runs the inliner
2.1.0	12/3/2008	First Draft	Irem Aktug (KTH)	Minor changes
2.2.0	13/3/2008	Final	Irem Aktug (KTH)	Minor changes

Executive Summary

This report documents the inliner prototype for java bytecode programs. It describes:

- how to download, install and use the tool and
- how to access an online demo.

Contents

1 Introduction

This document describes how to install and run the inliner program developed at KTH. The inliner inputs an arbitrary program in java bytecode, a policy written in ConSpec and information about the security relevant parts of the API and outputs a modified version of the program that respects the policy. ConSpec is a policy language developed in the course of S3MS. Further information about ConSpec including its syntax and its semantics as security automata can be found in [1].

2 Requirements

The only requirement for running the inliner is to have JRE. WTK (Sun Java Wireless Tool Kit for CLDC) should be installed for packaging the modified application (the output of the inliner), so that it runs on a mobile device.

3 Obtaining and Installing the Inliner

The inliner consists of the file Inliner.jar and can be downloaded from the following url:

`http : //csc.kth.se/irem/S3MS/Inliner/downloads/Inliner.jar`

The downloads directory also contains the library files for API commonly used in mobile applications (cldcapi10.jar,cldcapi11.jar,midpapi10.jar,midpapi20.jar,midpapi21.jar,wma20.jar). An alternative for obtaining relevant API archive files is to have WTK (Sun Java Wireless Tool Kit for CLDC) installed in your system. The libs directory of this program includes archive files for various mobile API.
There is no installation procedure for the inliner. It is sufficient to save the file Inliner.jar to the same directory as the untrusted application.

4 Input to the Inliner

The following files are input to the inliner:

- Policy file: The policy should be written in ConSpec and should have scope `Session`.
- Application files: All .class files of the untrusted application should be input to the inliner. Notice that a mobile application is usually packed in a .jar file. Such an application can be input to the inliner as it is or after unpacking (using the command jar xvf *jarfile*).
- Library files: The archive (.jar) file for each API that is used in the application should be present in the system.

The directory of the tool contains sample files for the above, at the end of this document we give information on how one can use the above files to demo the execution of the tool.

5 Running the Inliner

The untrusted application should be placed in a subdirectory of the directory where the inliner is saved. The policy file should be placed in the same directory with the inliner.
The inliner is executed by running the following command at the directory where the inliner resides:
java -jar Inliner.jar -lib *libfile* [-lib *libfile* ...] -policy *policyfile* -target *targetfile* [-target *targetfile* ...]
The *policyfile* is a text file containing a single ConSpec policy with scope Session. The *libfile*s are all those .jar files that contain the declaration of a method used in one of the *targetfile*s. The *targetfile*

can be a .jar file or a .class file. If the class files are used, all class files of the application should be input to the inliner.
For instance, for inlining the application in MailApp.jar stored in the same directory as the inliner, for policy saved in file NoSMS.pol which mentions methods of the CLDC 1.1. API, we execute the following command:

```
java -jar Inliner.jar -lib cldcapi11.jar -policy NoSMS.pol -target MailApp.jar
```

6 The Working of the Tool

The inliner first reads the policy file and, if it is a valid ConSpec file, generates code for each event clause in the policy. The policy file is read into an abstract syntax tree with the help of code generated by the parser generator CUP (http://www2.cs.tum.edu/projects/cup/) and lexer generator JFlex (http://jflex.de/). After the syntax tree has been generated, it is traversed using the visitor pattern. Type checking and register allocation is performed as nodes are visited. A symbol table is maintained during this phase, but is then discarded, because the type information and allocated register indices (for local variables) is stored directly in the corresponding nodes of the syntax tree.
The next step is to "compile" the policy, i.e. create a mapping from event clauses to java bytecode. At runtime, the guards of an event clause should be evaluated from top to bottom, and the code for the first guard that is true should be executed. If no guard is true, the program violates the policy and an error message should be printed followed by termination of the program. The code for the guarded commands of the policy can then be generated by evaluating the code for each guard in turn, jumping to the next guard if the previous was false, and having code to terminate the program after the last guard. Directly after the evaluation of each guard, the code for the body of the guarded command is generated, followed by a jump past the termination code. The program is terminated by invoking the System.exit() method.
Each method of each class to be inlined is disassembled. The instructions are then visited in turn until an invoke instruction relevant to the policy is reached and it is replaced by the suitable instructions to enforce the policy. When all instructions of a method have been visited, the code for the method is assembled again. The reading and writing of class files is done with the ASM bytecode manipulation framework (http://asm.objectweb.org/).
Finally, the security state class is generated. It is a public class containing only public static fields corresponding to the security state variables of the policy. The code to initialize those fields is placed in the static initialization method which is automatically invoked by the JVM when the class is loaded.

7 Using the Output of the Inliner

The inliner first creates the directory working and uncompresses both application and library files into this directory. The application and library files are scanned before the program does any inlining, to build a class hierarchy table in program's memory. The application files are also used when producing the inlined code.
The inliner creates the directory inlined/working/ as a subdirectory of the current directory. The rewritten class files of the untrusted application are copied to this directory, preserving the directory structure of the original files. These are placed under a directory named after the application, for the above MailApp.jar the inlined program would be placed at inlined/working/MailApp/. Additionally, this directory contains the new class file SecState.class which is used to keep the security state as the modified program executes.
In order to run the modified application, the contents of the directory inlined/working/*appname/* should be first be verified using the verifier of the WTK. Assuming that the executables of WTK are

in the user's classpath, the following command can be used for this purpose:

```
preverify -classpath classpath -d targetdir inputdir| classfile [ classfile ... ]
```

The *classpath* should contain .jar files for all API used in the program, the same files input to the inliner as libraries. A safe approach for mobile applications is to include all .jar files found in the WTK library. The "-d" option places the verified files to the target directory indicated by *targetdir*. The *inputdir* is the directory that the inlined classes are in. In this case, it would be a directory ending with inlined/working/*appname/*. It is crucial to give the path to the root directory of the application, i.e. to include *appname*.

```
preverify -classpath cldcapi11.jar -d verifiedClasses inlined/working/MailApp
```

The final step for running the modified program is to recompress the verified files to a jar archive. This can be done by executing the following command at the root directory of the modified (and verified) application:

```
jar cvfm jarfile META-INF/MANIFEST.MF *
```

The produced jar file can then be run on a mobile device after updating the corresponding jad file to reflect the new size of the jar file.

8 An Online Demo

An online demo is displayed on the following page:
`http://www.csc.kth.se/irem/S3MS/Inliner/`

References

[1] I. Aktug and K. Naliuka. ConSpec – a formal language for policy specification. In F. Piessens and F. Massacci, editors, *Proc. of The First International Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM'07)*, volume 197-1 of *Electronic Notes in Theoretical Computer Science*, pages 45–58, Dresden, Germany, 2007