Privacy Leaks in Java Classes

Jacob Sparre Andersen
Research & Innovation, Copenhagen, Denmark

May 2014

One of the changes from DO-178B to DO-178C is supposedly to make it allowable to implement flight control systems in the Java programming language [2]. This makes it important to learn of the reliability deficiencies of Java, and possibly how to avoid them.

This presentation is focused on one particular class of programming errors in Java; **privacy leaks** from encapsulated data structures.

In Java assignment (=) works differently for simple and composite types (classes); simple type objects are copied while composite types have their reference copied.

One of the consequences of this design decision in Java is that "getters" for composite type attributes have to be written differently than those for simple type attributes. If one uses the simple type "getter" pattern for a composite type attribute, the result will be a privacy leak, where the user of the class will have access to modify a private attribute of the class directly – something which is in constrast with the whole idea of private attributes.

As an external examinator for Danish software engineering schools, I noticed that it seems to be difficult for students to grasp the practical consequences of the different handling of simple and composite types in Java. This has inspired me to look into how widespread privacy leaks are in "industrial" Java software.

This mistake seems to be quite common – even the Eclipse [1] "getter" generator makes the mistake – so I've put together a tool for identifying privacy leaking "getters" in compiled Java classes. The tool uses a Java decompiler to generate a normalised form of the source code for the Java classes, and then uses simple pattern matching to identify uses of the simple type "getter" pattern for composite, private attributes.

One of the five goals in the creation of the Java programming language is that it should be "robust and secure". The extent of privacy leaks in real-life Java classes indicates that there is still quite a way to that goal.

The presentation will

- discuss potential bugs derived from privacy leaks;
- show examples of safe and privacy leaking "getters";
- demonstrate how Eclipse generates unsafe source code;
- show the pattern searched for by the privacy leak tracker;
- reveal an interesting, real-life example of a privacy leaking Java class; and
- finally discuss why I don't consider this a similarly serious issue in Ada.

References

[1] Eclipse Foundation Inc. Eclipse - The Eclipse Foundation open source community website, 2014.

[2] Guy L. Steele Jr. James Gosling, Bill Joy and Gilad Bracha. *The Java Language Specification.* Addison-Wesley, 3rd edition, 2005.