BANANA
A Tool for Boundary Ambients Nesting ANAlysis

Chiara Braghin[1], Agostino Cortesi[1], Stefano Filippone[1], Riccardo Focardi[1], Flaminia L. Luccio[2], and Carla Piazza[1]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia,
{braghin,cortesi,sfilippo,focardi,piazza}@dsi.unive.it
[2] Dipartimento di Scienze Matematiche, Università di Trieste,
luccio@dsm.univ.trieste.it

1 Introduction

BANANA is a tool for the analysis of information leakage in mobile agent specifications. The language considered is Mobile Ambient calculus, initially proposed by Cardelli and Gordon with the main purpose of explicitly modeling mobility [5]. Sites and agents (i.e., processes) are modeled as nested boxes (i.e., ambients), provided with capabilities for entering, exiting and dissolving other boxes. This specification language provides a very simple framework to reason about information flow and security when mobility is an issue [1].

The main features of BANANA are:

- A textual and graphical editor for Mobile Ambients, to specify and modify the process by setting ambient nesting capabilities and security attributes in a very user-friendly fashion.
- A parser which checks for syntax errors and builds the syntax tree out of the Mobile Ambient process.
- An analyzer which computes an over approximation of all possible nestings occurring at run-time. The tool supports two different control flow analyses, namely the one of Nielson et al. [6] and the one by Braghin et al. [1].
- A post-processing module, that interprets the results of the analysis in terms of the boundary-based information-flow model proposed in [1], where information flows correspond to leakages of high-level (i.e., secret) ambients out of protective (i.e., boundary) ambients, toward the low-level (i.e., untrusted) environment.
- A detailed output window reporting both the analysis and the security results obtained by the post-processing module, and some statistics about the computational costs of the performed analysis.

BANANA is implemented in Java and strongly exploits the modularity of object-oriented technology, thus allowing scalability to other analyses (e.g., the one in [3]) and extensions of the target language (e.g., [7]). Moreover, BANANA is conceived as an applet based on AWT and thus compatible with the majority of current web browsers supporting Java.

2 Security of Mobile Ambients

The Mobile Ambient calculus has been introduced in [5] with the main purpose of explicitly modeling mobility. Indeed, ambients are arbitrarily nested boundaries which can move around through suitable capabilities. The syntax of processes is given as follows, where $n \in \mathbf{Amb}$ denotes an ambient name.

$$
\begin{array}{llllll}
P, Q ::= & (\nu n)P & \text{restriction} & \mid\; n^{\ell^a}[\![\, P \,]\!] & \text{ambient} \\
\mid & \mathbf{0} & \text{inactivity} & \mid\; \mathbf{in}^{\ell^t}\, n.P & \text{capability to enter } n \\
\mid & P \mid Q & \text{composition} & \mid\; \mathbf{out}^{\ell^t}\, n.P & \text{capability to exit } n \\
\mid & !P & \text{replication} & \mid\; \mathbf{open}^{\ell^t}\, n.P & \text{capability to open } n
\end{array}
$$

Intuitively, the restriction $(\nu n)P$ introduces the new name n and limits its scope to P; $P \mid Q$ is P and Q running in parallel; replication provides recursion and iteration. By $n^{\ell^a}[\![\, P \,]\!]$ we denote the ambient named n with the process P running inside it. The capabilities $\mathbf{in}^{\ell^t}\, n$ and $\mathbf{out}^{\ell^t}\, n$ move their enclosing ambients in and out ambient n, respectively; the capability $\mathbf{open}^{\ell^t}\, n$ is used to dissolve a sibling ambient n. Labels on ambients and on transitions are introduced as it is customary in static analysis to indicate "program points".

The operational semantics of a process P is given through a suitable reduction relation $\rightarrow$ and a structural congruence $\equiv$ between processes. Intuitively, $P \rightarrow Q$ represents the possibility for P of reducing to Q through some computation (see also [5]).
For instance, let P_1 be a process modeling an $envelope$ sent from $venice$ to $warsaw$:

$$venice[\![\; envelope[\![\, \mathbf{out}\, venice.\mathbf{in}\, warsaw.\mathbf{0} \,]\!] \mid Q \,]\!] \mid warsaw[\![\, \mathbf{open}\, envelope.\mathbf{0} \,]\!]$$

Initially, $envelope$ is in site $venice$. Then, it exits $venice$ and enters site $warsaw$ by applying its capabilities $\mathbf{out}\, venice$ and $\mathbf{in}\, warsaw$, respectively. Once site $warsaw$ receives the $envelope$, it reads its content by consuming its $\mathbf{open}\, envelope$ capability. At this point, process P_1 has reached the state: $venice[\![\, \mathbf{0} \,]\!] \mid warsaw[\![\, Q \,]\!]$.

To deal with security issues, information is classified into different levels of confidentiality. This is obtained by exploiting the labelling of ambients. In particular, the set of ambient labels is partitioned into three disjoint sets: *high*, *low* and *boundary* labels. Ambients labelled with boundary labels (*boundary ambients*) are the ones responsible for confining confidential information. Information leakage occurs if a high level ambient exits a boundary, thus becoming possibly exposed to a malicious ambient attack. For instance, let P_2 be an extension of process P_1, in which the $envelope$ contains confidential data $hdata$ (labelled $high$) which needs to be safely sent from $venice$ to $warsaw$.

$$
\begin{array}{l}
venice^{b_1}[\![\; envelope^{b_2}[\![\, \mathbf{out}^{c_1}\; venice.\mathbf{in}^{c_2}\; warsaw.\mathbf{0} \mid\; hdata^{h}[\![\, \mathbf{in}^{c_3}\; translator.\mathbf{0} \,]\!] \,]\!] \,]\!] \mid \\
warsaw^{b_3}[\![\, \mathbf{open}^{c_4}\; envelope.\mathbf{0} \,]\!] \mid\; translator^{m}[\![\, \mathbf{in}^{c_5}\; envelope.\mathbf{0} \,]\!] \mid \mathbf{open}^{c_6}\; translator
\end{array}
$$

In this case, $venice$, $warsaw$ and $send$ must be labelled $boundary$ to protect $hdata$ during the whole execution. (See [1, 2] for more detail.)

The tool verifies that in every execution of process P no direct information leakage occurs, by implementing the control flow analyses described in [6] and [1]. Both analyses aim at modeling the possible nesting of processes occurring at run-time.

3 Tool Overview

A screen-shot of the BANANA tool is shown in Figure 1, while Figure 2 gives an overview of the architecture of the tool.

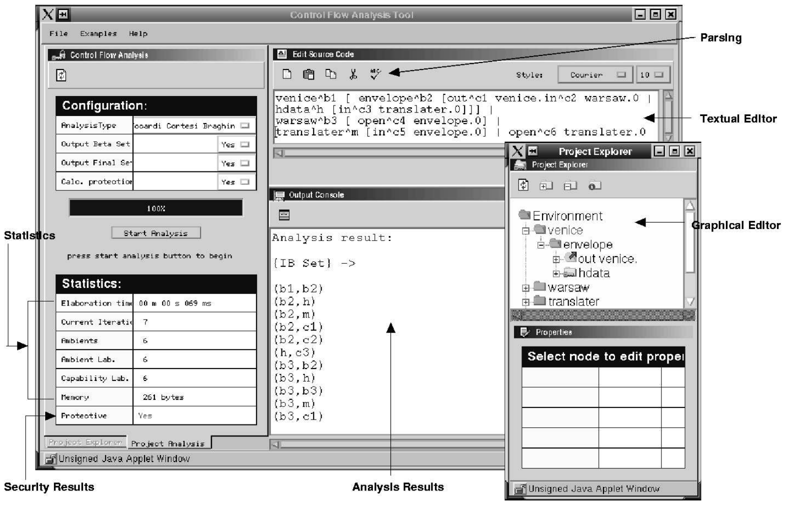

Fig. 1. Screen-shot of the BANANA tool.

A user can edit the process to be analyzed by using either the Textual or the Graphical Editor. The security labelling (i.e., the labels denoting untrusted, confidential, and boundary ambients) can be inserted directly by the user, or automatically derived by the tool during the parsing phase. In the latter case, ambients starting with letter 'b' are labelled boundaries, while ambients starting with 'h' are labelled high. By selecting an item in the Project Explorer window, the user can check/modify the properties of the ambient/capability. The user can also check the syntax correctness of the process by selecting the Parsing button.

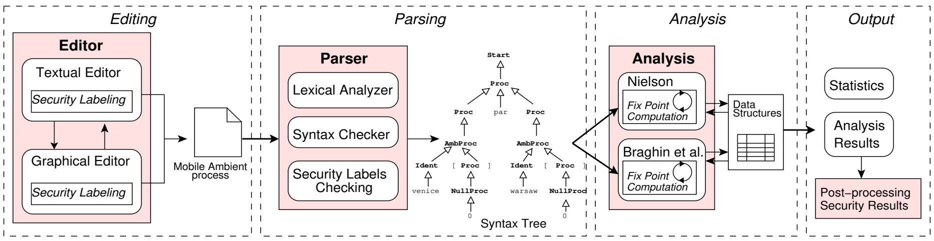

Fig. 2. Overview of the BANANA tool.

The user can then choose to launch either the Nielson or the Braghin et al. analysis. Once the analysis has started the tool parses the process, builds a syntax tree and computes the fix point algorithm yielding an over-approximation of all possible ambient nestings. The result of the analysis is reported in the Output Console as a list of pairs of labels.

By post-processing the analysis results, BANANA reports in the filed Protective the sure absence of information leakages.

The BANANA tool is available on-line at the following address:
`http://www.dsi.unive.it/~dbraghin/banana/`

4 Conclusion

BANANA is a very user-friendly tool for the analysis of information leakage in Mobile Ambient specifications.

It allows to compare two different control flow analyses, i.e., the one by Nielson et al. and the one by Braghin et al., run on non-trivial examples. This also shows the feasibility of these analyses, that have polynomial time and space complexities.

Another interesting issue is the scalability of this tool: as an on-going work we are currently extending BANANA to computationally more efficient analyses, as the one in [3] that has improved time and space complexities. This task is particularly simple, because of the modularity of the object-oriented Java technology.

The scalability is also possible toward variants of the Mobile Ambient calculus and extensions of the analysis, such as the one of [4] where a minimal set of security boundaries is inferred.

References

1. C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages*, Elsevier, to appear, vol. 18, 2002.
2. C. Braghin, A. Cortesi, and R. Focardi. Control Flow Analysis of Mobile Ambients with Security Boundaries. In B. Jacobs and A. Rensink, editors, *Proc. of Fifth Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'02)*, pages 197–212, Kluwer Academic Publisher, 2002.
3. C. Braghin, A. Cortesi, R. Focardi, F.L. Luccio, and C. Piazza A New Algorithm for Control Flow Analysis of Mobile Ambients. In *Proc. of The 4th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'03)*, LNCS, to appear, 2003.
4. C. Braghin, A. Cortesi, R. Focardi, and S. van Bakel. Boundary Inference for Enforcing Security Policies in Mobile Ambients. In *Proc. of The 2nd IFIP Int. Conf. on Theoretical Computer Science (TCS'02)*, pages 383–395. Kluwer Academic Publisher, August 2002.
5. L. Cardelli and A.D. Gordon. Mobile Ambients. Theoretical Computer Science (TCS), 240(1):177-213, 2000.
6. R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. In *Proc. of Static Analysis Symposium (SAS)*, volume 1694 of Lecture Notes in Computer Science, pages 134–148. Springer-Verlag, September 1999.
7. Francesca Levi and Davide Sangiorgi. Controlling Interference in Ambients. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 352–364, 2000.