Language Issues in Mobile Program Security

Dennis Volpano and Geoffrey Smith

summarized by Sean Davey

November 3, 1999

1 Introduction

In the environment of world-wide-networks inhabited by mobile agents, security is an increasingly important issue. While not limited to mobile computing, this paper discusses security issues in language design that are very pertinent to it. Having split the concerns into two domains, host security and code security, and having decided that code security is intractable because mobile code is actually controlled by the hosts, the paper focuses on host security.

2 Host Security

The two main issues covered by the paper are privacy and integrity. Privacy is the notion of some information contained in the host that may be needed by the agent to perform its function but that the host doesn’t want leaked out by the agent in any way. The author’s limit the privacy domain to those events which are observable within the host and are not including human observation online or other exterior observers. Integrity is the notion of code correctness or at least guarantees that the mobile agent cannot perform any damaging actions to the host. Traditionally integrity is enforced by security architectures that disallow unwanted actions by the agent. The authors feel that the increasing complexity of the security architectures that are coming out may be due to poor language design. In the future they expect security architectures to be less complex, more formally defined and rely more on the well-formedness of programs to come from the language design.

3 Type Safety

Type safety is the notion of preventing interpretation of data as something that it’s not. This includes not allowing user defined objects to represent themselves as system objects which may have more privileges.
3.1 Type Preservation

Type preservation exists if expressions produce values of the correct type. Problems still exist even with type preservation. For example, dereferencing a dangling pointer is possible in C. The definition of C as a language say that programs that do this are unpredictable whereas a more secure language that allows dangling pointers would provide specifications of how a program should perform if a dangling pointer is dereferenced. Type preservation also does not prevent mobile code from obtaining references to system security objects as was possible in Java for JDK 1.1.1\(^1\) and which severely undermined its trust framework.

3.2 Type Soundness

A type soundness theorem specifies all the possible ways a program can behave. A type system is sound if it is not required to catch any runtime errors that cause a program to abort. Thus the soundness criteria is determined by the initial expectations of which errors are reasonable to catch. Type soundness can be defined as a property of a type system and can be used a means of comparing such systems. Although Java avoids many of the possible type and other integrity problems found in C, it is still difficult to statically analyze java byte code in order to make security guarantees. In particular this is difficult because Java allows for self-modifying code by allowing classes to be dynamically constructed.

4 Privacy

In a host environment, some data accessible by the agent may be considered private. Ensuring that the agent does not leak any of this data can be difficult. An obvious explicit way to leak data is to assign a private variable to a public one. There are many more methods which are difficult to detect. The authors call these methods *implicit channels*. An example is a loop that checks bits of a private variable with a mask and then sets bits of a public variable using with the mask. To prevent this Termination Security is introduced which states that changing the contents of private variables cannot influence the outcome of the public variables. A possible implicit channel might check bits of a private variable and loop forever to indicate a zero. Thus the private data can be determined through multiple runs of the agent even though the agent doesn’t violate Termination Security. So, Offline Security is introduced which states that changing the contents of a private variable cannot influence the outcome of the public variables nor change the programs normal termination. A possible implicit channel can get around this by dividing by zero to indicate the value of a bit of a private variable. Online Security is basically Offline Security extended to include the independence of a program’s abnormal termination and its time until termination. The opportunities for leaking data are dependent on what the

\(^1\)In JDK 1.1.1, applications can obtain a reference to the code signers array
agent can observe about the system. The author’s suggest that Online Security maybe to restrictive for mobile agents and that some of the problems can be better solved by changing what is observable to the agent. Even more implicit channels are possible in a threaded system. For example two threads could communicate bits of a private variable using busy-wait loops as a semaphore.

5 Formal Logic of Privacy

The final section of the paper presents formal specification for different rules that ensure the different type soundness properties. If a given language has a desirable type soundness then run-time type checking is no longer necessary. However, static analysis is required to find many of the implicit channels. The authors are currently implementing some of these ideas in a code verification pipeline for Java. Analyzing Java byte code is more difficult as language structures are not as readily apparent. Complete static analysis while decidable is not recursively enumerable. Therefore completeness is often compromised but it's an open question how often constructs that are not not caught by the incomplete analysis occur.

6 Conclusion

The authors summarize their paper which mostly focused on type safety and privacy. They also advise language designers to consider security issues from the beginning. The need for formal semantics is stressed. They talk about the static analysis being built into a compiler but I found this surprising because that solution won’t work in mobile computing. Finally they complain that threads make everything far too difficult. In my opinion this paper points out some interesting issues and does a good job in discussing type soundness formally. However I don’t really feel that the paper was well directed, some of the conclusions seemed a bit strange and out-of-place and some important issues were ignored.

---

2For example, making the host time stamp messages sent by the agent instead of letting the agent see the clock itself.