MOBILE CODE AND SECURITY

Gary McGraw
Reliable Software Technologies
Edward W. Felten
Princeton University

Running somebody else's code on your computer is a risky activity. It's an old problem, but it gets a new twist on the World Wide Web.

Mobile code is code that traverses a network during its lifetime and executes at the destination machine. In its most powerful guise, the same piece of mobile code is able to run on multiple platforms (both Unix boxes and Win32 machines, for example). This powerful idea opens up many new possibilities on the World Wide Web, a heterogeneous collection of machines. For the first time it is possible to have Web-based programs whose behavior is coded in some common programming language. This code need not be compiled to tens of platforms and distributed only after determining the target platform. Instead, mobile code is written once and then runs wherever it ends up. There are many well-known systems for creating and using mobile code, including Java, JavaScript, VBScript, ActiveX, Postscript, and Word macros.

This special issue of *IEEE Internet Computing* is concerned with the security implications of mobile code. The problem is simply this: Running somebody else's code on your computer is a risky activity. It is not a new problem, of course. It's an old problem with a new twist.

A BRIEF HISTORY OF SECURITY CONCERNS FOR MOBILE CODE

In the early days of the Internet, everyone agreed that downloading arbitrary binaries and executing them on your machine was a Bad Idea. Of course, most people did it anyway, and some people suffered as a result. By the mid 1980s, there was a lot of freeware and shareware available to download. To find it, you could use Archie, which provided a way to search a large index of anonymous FTP content. Once you found some leads (often several ASCII pages worth), you chose your target and FTP'd what you needed. Then you installed and ran it.

The risks of downloading and running a random person's code on your machine are clear. If the code has a virus attached, it can infect your machine. If the program is a Trojan Horse, it can take over your machine for its own purposes while appearing to do something useful. How can we be sure that a program someone says is useful hasn't been hijacked to do something nasty?

1089-7801/98/$10.00 ©1998 IEEE

By the late 1980s, checksumming provided part of the answer. A checksum is a simple computation performed on a piece of code to provide a digest or a "thumbprint" of the code using a one-way hash function. At least a few anonymously downloadable programs included checksum data that could be used for rudimentary data integrity purposes, helping to ensure that the code downloaded at the end of the communication pipe was the same as the original code distributed by its author. Of course, who was to say that a program's checksum hadn't been tampered with or that an author did not include Trojan Horse functionality?

The Web got its start in 1992. At first the HTML-based Web was static. Mobile code systems like Java and JavaScript changed all that, making it possible for a Web server to provide programs as content. The dangers of mobile code and systems for addressing these dangers are the focus of this issue. The problem is that systems like Java and JavaScript are so unobtrusive that a Web user may not even notice that they are requesting and running mobile code on their computer. What once required a conscious decision and several steps in a technological process now requires a simple click of the mouse on a Web link.

This extra risk is not very alarming if we operate under the assumption that the code we're running is trusted. But why should a Web user trust a random piece of code encountered on the Web? Clearly they should not. As a reaction to this concern, many popular forms of mobile code (including Java) were designed to run *untrusted* code (and, more recently, *partially trusted* code) in a secure manner. This issue discusses how mobile code systems attempt to do this and to what extent they succeed.

DEVELOPER CONCERNS

From a technology perspective, a complete security solution for mobile code will always involve two distinct parts. First, the mobile code platform itself must be secure. For example, unless the Java Virtual Machine is doing its job perfectly, Java's language-based approach to security will not work.[1] Second, any code developed for use on the mobile code platform must be designed and implemented with security in mind. In other words, simply writing code in Java is no guarantee of secure behavior.

Obviously, developers play a critical role in both parts of the solution. Creating a mobile code system is a complex undertaking that pushes the limits of available technology and research.

Consider Java. Sun Microsystems and the Java licensees have worked hard to evolve a system that supports secure creation and use of mobile code. But Java is not immune to security risks.[1,2] Implementing a language-based security model is not easy, and there are bound to be mistakes.

A prime example of the ways in which Java is pushing the envelope is its combination of type safety and dynamic loading. Java bases its security approach on type safety. Programs must be prevented from accessing memory and other security-critical resources in inappropriate ways. Every piece of memory is part of some Java object, including the classes that make up the security model itself. Java is also designed so that classes can be dynamically added and removed from the runtime environment. Dynamic loading is handled by Java's class loading mechanisms. The question is how to sustain type safety and allow dynamic loading at the same time. This complex question has been addressed in a recent dissertation.[3] Many open research issues remain.

What once required a conscious decision and a process now requires only a mouse click.

Programming language researchers have also done some work trying to prove the soundness of the Java language and VM definition.[4,5] Although this work is still in preliminary stages, some interesting and suggestive results are available. The bottom line is that the definition of Java will probably turn out to be sound, once many more details have been filled in.

Developers are charged with creating secure implementations of both the mobile code platform and the code that eventually runs on the platform. Both of these tasks require a developer to navigate the often uncharted waters of language-based security. A necessary side effect of implementing mobile code (which can in some ways be considered a very large security-related experiment) is making mistakes and learning from them.

What we have learned is that it is important to develop mobile code and mobile code platforms with a defensive stance so that neither is susceptible to attack. Unfortunately, doing so requires extensive knowledge both of particular mobile code systems and of computer security issues. In our book, *Securing Java*, we list 12 rules Java developers should follow so that their code is harder to attack.[1] Similar rules can be developed for other mobile code systems.

You can make gaffes developing security-critical code in any language.

Developing secure programs in any language is neither trivial nor automatic. Anybody who reads the newspapers or the trade press can see how often skilled programmers write code with security bugs. You can make gaffes developing security-critical code in any language. Know your enemy. Think about what might confront your code in terms of malicious attacks. Mitigate risks by designing, coding, and testing carefully.

THE ARTICLES

Through a peer-review process, we selected four articles that touch on the central issues of mobile code security.

Two of them provide broad views of this domain.

"Mobile Code Security," (pp. 30-34) by Avi Rubin and Dan Geer, introduces the fundamental issues and outlines five different approaches to securing mobile code: the Java sandbox as implemented in the Java Developers Kit 1.0 and 1.1, code signing as found in Authenticode and the diverse Java code signing systems of JDK 1.1, the combination of sandboxes and signatures as found in JDK 1.2, firewalling, and proof-carrying code. The authors give excellent citations to more in-depth discussions of each approach.

"Securing Systems Against External Programs," (pp. 35-45) by Brant Hashii, Manoj Lal, Steven Samorodin, and Raju Pandey, describes and categorizes security risks associated with mobile code. The security models introduced by Rubin and Geer are discussed in light of these risk categories. Solutions are classified according to a simple framework of phases encountered during the lifetime of mobile

WHERE TO LEARN MORE ABOUT MOBILE CODE SECURITY

The references of each of the four articles included in this issue are all good places to begin further inquiry. In addition, the following resources may be useful.

PAPERS

M. Erdos, B. Hartman, and M. Mueller, "Security Reference Model for the Java Developers Kit 1.0.2," white paper, Sun Microsystems, 1996, http://www.javasoft.com/security/SRM.html.

D. Hopwood, "A Comparison Between Java and ActiveX Security," available online at http://www.users.zetnet.co.uk/hopwood/papers/compsec97.html.

D. Martin, S. Rajagopalan, and A. Rubin, "Blocking Java Applets at the Firewall," *Proc. 1997 Network and Distributed System Security Symp.*, Internet Society, 1997; available at http://www.cs.bu.edu/techreports/96-026-java-firewalls.ps.Z.

G. Necula and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," *Proc. Second Symp. Operating Systems Design and Implementation*, Usenix Assn., Berkeley, Calif., 1996, pp. 229-243. (Proof-carrying code.)

D. Wallach, *A New Approach to Mobile Code Security*, doctoral dissertation, Princeton Univ., Dept. of Computer Science, 1998.

WEB SITES

The Java Security Hotlist (over 100 links in nine categories) • www.rstcorp.com/javasecurity/links.html

Princeton's Secure Internet Programming Laboratory (see especially the Publications and FAQs) • www.cs.princeton.edu/sip/

Mobile Code Bibliography (includes some security-related papers) • www.cnri.reston.va.us/home/koe/bib/mobile-abs.bib.html

Mobile Code Security Bibliography • fas.sfu.ca/cs/people/GradStudents/pwfong/personal/Security/

University of Washington's Kimera group • kimera.cs.washington.edu/

JavaScript Security Flaws • www.osf.org/~loverso/javascript/

code: development, migration, and execution. Diverse approaches for securing mobile code are compared and contrasted.

The third and fourth articles describe models related to securing Java code and code written in Web scripting languages.

"Secure Web Scripting," (pp. 46-55) by Vinod Anupam and Alain Mayer, begins with a short description of the security risks associated with simple languages for embedding scripts in the HTML of a Web page for interpretation by a browser. A security design is proposed that can prevent some of the attacks commonly encountered on the Web. Many of the fundamental characteristics of the proposed design are borrowed directly from secure operating systems research.

"Secure Java Class Loading," (pp. 56-61) by Li Gong, describes Java's class loading mechanism, an essential piece of the Java security model. Java's language-based approach to security is an enforcement model based on four pieces: the *verifier*, which provides some type safety analysis; *class loaders*, which add and remove classes dynamically from the Java runtime environment, a *security manager* that enforces restrictions on untrusted code and is the most apparent piece of the security model, and (new to JDK 1.2) an *access controller*, which uses stack inspection and codified security enforcement rules (specified by the user) to make access control decisions. Java's dynamic class loading capability is hard to reconcile with the security requirement for type safety. After all, if code has not yet arrived, how can we know that it is being used in a type-safe manner? Java's sophisticated class loading approach helps address this complex issue. Gong's article discusses the new JDK 1.2 design with an emphasis on the internals of class loaders.

CAN MOBILE CODE BE SECURE?

Today's diverse approaches to securing mobile code are all works in progress. Each different implementation of mobile code, including Java, ActiveX, and JavaScript, faces similar security risks; but each system presents a different way of dealing with the risks. In our opinion, Java's security design stands head and shoulders above the competition. But Java is a complex system that is evolving at a serious clip. Securing Java and other forms of mobile code is still as much an art as it is a science.

There is no such thing as absolute security, whether the code is mobile or stays put right where it is developed. Creating and maintaining secure systems requires rigorous software assurance and the ability to manage risks wisely. Mobile code makes securing a system more complicated, but the benefits of mobile code are often worth a bit of added risk. As the world becomes increasingly networked and interdevice communication becomes essential, mobile code is likely to play an increasingly important role. ■

REFERENCES

1. G. McGraw and E. Felten, *Securing Java: Getting Down to Business with Mobile Code*, John Wiley and Sons, New York, N.Y, 1998.
2. D. Dean, E. Felten, and D. Wallach, "Java Security: From Hotjava to Netscape and Beyond," *Proc. 1996 IEEE Symp. on Security and Privacy*, IEEE Computer Society, Los Alamitos, Calif., 1996.
3. D. Dean, "Formal Aspects of Mobile Code Security," doctoral dissertation, Princeton Univ., Dept. of Computer Science, 1998.
4. S. Drossopoulou and S. Eisenbach, "Towards an Operations Semantics and Proof of Type Soundness for Java," tech. report, 1998; available at http://outoften.doc.ic.ac.uk/projects/slurp/papers.html.
5. R. Stata and M. Abadi, "A Type System for Java Bytecode Subroutine," *Proc. 25th ACM Symp. on Principles of Programming Languages*, ACM Press, New York, 1998, pp. 149-160.

Gary McGraw is vice president of business development at Reliable Software Technologies. He holds a dual PhD in cognitive science and computer science from Indiana University and a BA in philosophy from the University of Virginia. McGraw coauthored both *Java Security: Hostile Applets, Holes, and Antidotes* (John Wiley & Sons, New York, 1996) and *Securing Java: Getting down to Business with Mobile Code* (Wiley, 1998) with Ed Felten. He has published over 50 peer-reviewed technical papers and is principal investigator on grants from Air Force Research labs, DARPA, and NIST's Advanced Technology Program. McGraw is a member of the IEEE, AAAI, and Cognitive Science Society.

Edward W. Felten is assistant professor of computer science at Princeton University. His research specialties include operating systems, Internet software, and Internet security. He received his BS with honors in physics from the California Institute of Technology in 1985 and his PhD in computer science and engineering from the University of Washington in 1993. He has published more than 40 research papers and won both a National Science Foundation Young Investigator award and an Alfred P. Sloan fellowship for his work.

Readers may contact McGraw and Felten via email at gem@rstcorp.com and felten@cs.princeton.com, respectively.