



Information System Security Operation Self-Protecting Mobile Agents

Protecting Mobile Agents from Malicious Host Computers

Overview

The objective of the Self-Protecting Mobile Agents (SPMA) project was to develop practical and effective techniques to allow mobile agents to protect themselves from malicious host computers. To speed the project and to maximize technology transfer, we based our work on Open Source systems, and are releasing our tools under Open Source license terms.

We built tools that allow us to operate on Java object programs to investigate, obfuscate, and add features to them without access to agent source code.

We also developed tools to translate an individual software agent into a distributed set of tamper-resistant agentlets that is never entirely vulnerable to a single host, and that can detect and recover from compromise of a subset of its elements.

Approach

Our initial plan was to develop tools that translate an individual software agent into a distributed set of tamper-resistant agentlets that is never entirely vulnerable to a single host, and that can detect and recover from compromise of a subset of its elements. We planned to provide strong protection by combining three core techniques: Distributed Agent State; Obfuscation with Periodic Regeneration; and Monitoring and Recovery.

- Distributed Agent State. Agents can be partitioned into a set of communicating programs (agentlets) executing on independent hosts. Critical information is spread across the agentlets, thus limiting vulnerability to any proper subset of the hosts.

- Obfuscation with Periodic Regeneration. Executable code and data of each agentlet can be obfuscated using a variety of techniques (e.g., randomly selected, but equivalent, algorithms and data representations). We planned to use obfuscation to ensure that a host that chooses to execute an agent cannot, for a period of time, subtly modify the agent. Obfuscation was expected to delay, but not prevent, corruption or brainwashing of agents using reverse engineering. We planned to extend this limited protection by dynamically generating newly obfuscated versions of agents, and removing older versions from service.

We intended to set our regeneration periods so that a successful attack on an agentlet cannot be accomplished before the agentlet expires, using regeneration information from multiple agentlets (hosts), and hence not vulnerable to reverse-engineering by any single host.

- Monitoring and Recovery. We planned to make agentlets self-monitoring and have them monitor other agentlets. Using challenge/response techniques, agents would automatically exclude compromised agentlets, report the identities of tampering nodes, and replace lost agentlets.

In the first phase, we developed source-code translation tools to convert an individual software agent into a set of replicated communicating agentlets, and demonstrated a group of agentlets detecting and regenerating missing members.

In the second phase, we built a tool, the Java Binary Enhancement Tool (JBET), that reads and writes Java object files. JBET analyzes a set of class files and supports an extensible set of operators that can modify the internal tree. We built powerful obfuscation operators that ran

This work sponsored by DARPA through SPAWAR, Contract Number N66001-00-C-8602 with McAfee Research, which is now the Security Research Division of SPARTA.



Self-Protecting Mobile Agents

Protecting Mobile Agents from Malicious Host Computers

under JBET and demonstrated our ability to automatically obfuscate groups of Java classes.

When we considered how often we would have to re-obfuscate, we realized that we had no way to prove that any obfuscation method would resist reverse-engineering for a known minimum period of time. With the agreement of our sponsors, we redirected our project to producing a study that provides a stronger understanding of obfuscation.

Accomplishments

Tool Development. We further developed the Java Binary Enhancement Tool (JBET). JBET has a modular architecture that allows us to write drop-in code transformation operators that operate on the semantic program tree and write out Java class files representing the modified program. We implemented several code obfuscation transforms and evaluated their effectiveness.

JBET is written completely in Java, and packaged with a Makefile and an automated test suite. During our development, we instituted a nightly build that checked out the source, rebuilt JBET, ran the test suite, and reported results automatically. The tests included obfuscating JBET itself and ensuring that the obfuscated JBET produced the same results as the unobfuscated version.

Demonstrations. In demonstrations of our research, we obfuscated a simple program, and a program that did DES encryption, and showed that the output was noticeably harder to understand, at the cost of being much larger and slower than the input.

Technical Report. We wrote and released "Self-Protecting Mobile Agents Obfuscation Techniques Evaluation Report," TIS Report #01-036, describing the range of possible obfuscation techniques and their realization in JBET. This report presents an analysis of various program obfuscation techniques, describes JBET, and explores the real-world tradeoffs in translating and obfuscating Java bytecodes. We considered methods for obfuscating long-term and temporary data values, control flow, memory management, and type representation. We presented techniques with considerably more power than other available obfuscators.

In our report, we conclude that there is currently no evidence supporting the use of obfuscation for program protection. Both theory and experiment show that obfuscation alone cannot provide an assured level of security.

Obfuscation Research. As we progressed further with SPMA development, it became clear to us that the viability of our approach depended on being sure that an obfuscated program cannot be de-obfuscated for some amount of time. Also, we had no basis for estimation of the strength of obfuscation provided by any technique. We redirected our work to study the strength of obfuscation.

In search of a practical way to estimate the work factor involved in de-obfuscating a program obfuscated with different methods, we studied the theoretical basis for obfuscation. The paper by Barak, et al., titled "On the (Im)Possibility of Obfuscating Programs," presented at Crypto 2001, showed that "perfect obfuscation" is impossible. A consequence of their theorems is that there can be no universal method of computing the minimum time required for de-obfuscation. We examined their results to see if they hold for more practical obfuscation definitions.

We also built de-obfuscation tools using JBET, and found that it was easy to recognize and de-obfuscate programs we had obfuscated. We were able to find de-obfuscation methods for many other proposed methods of obfuscation.

Technology Transition. JBET is a useful tool for the analysis of Java bytecodes. It can be used for a variety of mobile-code security purposes, and is not limited to agent applications or obfuscation transforms. JBET's program analysis abilities were used in another DARPA research project, the Survivable Server project, to modify commercial Java binaries to add additional protection calls without access to the original source.

JBET and some example plugin operators have been released as open source software. Visit our Web page at <http://opensource.sparta.com>.

For more information call us at 410-872-1515, send an e-mail to ISSO-research@sparta.com, or visit us on the Web at <http://www.issosparta.com/research>.