The SeCode Approach: Towards Fault-Tolerant and Secure Execution of Mobile Code

Erica Y. Yang Jie Xu Keith Bennett
Department of Computer Science
University of Durham
South Road, Durham DH1 3LE, U.K.
Email: {Erica.Yang, Jie.Xu, Keith Bennett}@durham.ac.uk

Abstract

The dependability issue including fault tolerance and security is a basic stumbling block to the practical and commercial application of the mobile code technology. This short paper introduces the SeCode approach to fault-tolerant and secure execution of mobile code. The research focus is on the development of a method and an architectural framework to support mobile code against unintentional/intentional faults and malicious attacks from its operating environment. The proposed approach makes no assumption about the operating environment (i.e. remote hosts) for mobile code. It integrates work on fault tolerance and security within a well-defined formal system model, and offers a powerful ability to detect and identify faulty hosts and malicious attacks by means of redundant data structures with advanced fault diagnosis and cryptography techniques.

1. Introduction

Traditionally, security models, fault tolerance and security policies for large distributed systems have taken a "host-central" view so that the emphasis has been on protecting hosts, and making hosts reliable and fault-tolerant. This has worked very well for traditional client-server models, but is no longer adequate for envisaged approaches for the *next* generation of distributed computing based on mobile code. In a mobile code system, protection and fault tolerance must be provided for both hosts *and* mobile code. Especially, effective and feasible solutions must be developed to protect mobile code against malicious attacks from hosts.

With mobile code, we aim to move the code so that it is local to the associated resources needed. The gain is in terms of performance (locality), late binding, reconfiguration (resource location is not built into the application) and scalability. Specific application examples include mobile *e*-commerce, online purchasing and delivery, and Web access for retrieval of real-time information such as stock quotes, flight and reservation information, navigational maps, and weather reports. However, if mobile code is to be used for serious industrial applications, it is imperative that security and fault tolerance architectures are used a priori, otherwise users will not be able to trust the system [6].

There are in general two categories of research issues related to mobile code fault tolerance and security: i) protecting hosts against malicious and/or faulty mobile code, and ii) protecting mobile code against malicious and/or faulty hosts. Considerable efforts have been focused on the former problem, while the latter is still not well understood. Existing approaches and techniques for protecting code are limited to several separate areas such as tamper/failure detection, fault-tolerant execution and privacy preservation [2].

Fault-tolerant mobile code systems are usually based on replication and voting techniques to mask the effects of faulty or malicious hosts, and use cryptography techniques (e.g. digital signatures and the secret sharing scheme) to maintain the confidentiality of mobile code, authenticate their origin, and verify their integrity [3][4][9]. Most of existing solutions require the provision of system-level mechanisms (e.g. those supporting replication, reliable detection, and voting) at each remote host [5][7][8]. They generally do not scale well and thereby are not applicable in actual settings, such as the Internet.

In reality, the operating environment for mobile code is often large-scale and potentially non-trustworthy [6]. We have been developing a new approach, named SeCode, that does not require any system-level support for replication, detection and voting at any remote host. Our approach can detect corrupted results and reconstruct the expected results out of a threshold number of correct pieces of results, thereby tolerating faults and/or attacks from malicious remote hosts.

2. The SeCode Approach

Mobile code is defined here as executable code which is dispatched, which uses remote resources, and which reports back on its results. We consider a large-scale and dynamic network environment composed with a large number of hosts. Those hosts may provide high-level services, including information databases, interfaces to intelligent devices (e.g. sensors and display), and brokering services. Suppose that a local host (the origin) containing a user application needs remote resources. It therefore dispatches executable code to a set of m remote hosts, denoted by $H = \{H_1, H_2, \ldots, H_m\}$, to acquire the resources. Special services provided by our SeCode system will split the user request R at the local host into k

sub-requests, which are independent from each other. These sub-requests contain some redundancy so that *any* k - t out of k sub-requests are mathematically equivalent to the original request R (where t is the threshold number of faults and/or attacks). Instead of dispatching a single piece of mobile code, the system will send k pieces of the code, denoted by $C = \{C_1, C_2, \ldots, C_k\}$, that carry the sub-requests respectively to remote hosts. The expected results for the request R can be derived from the results returned as long as at least k - t sub-requests have been met, thereby effectively tolerating faults and/or attacks from malicious remote hosts.

The information resources (e.g. databases) at remote hosts can be modelled in different ways. We start with a simple model used in [1] and view the resources available at each remote host H_j as a binary string of length k_j. These strings are represented as follows:

$$
\begin{array}{l}
H_1\text{:}\quad x^1 = x^1{}_1 x^1{}_2 \ldots x^1{}_{k1}; \quad x^1{}_i \in \{0, 1\} \\
H_2\text{:}\quad x^2 = x^2{}_1 x^2{}_2 \ldots x^2{}_{k2}; \quad x^2{}_i \in \{0, 1\} \\
\quad\ldots\ldots \\
H_m\text{:}\quad x^m = x^m{}_1 x^m{}_2 \ldots x^m{}_{km}; \quad x^m{}_i \in \{0, 1\}
\end{array}
$$

The user application at the local host first creates an index set $i = \{i_1, i_2, \ldots, i_m\}$, where $i_j \in \{1, 2, \ldots, k_j\}$ and $j = 1, 2, \ldots, m$. The index i_j is used to indicate the user's interest in the specific value of the bit $x^j{}_{ij}$ at the host H_j. For the purpose of fault tolerance and security, the system also uses a set of random numbers, $r = \{r_1, r_2, \ldots, r_m\}$ where r_j is of length £^{rj} and $j = 1, 2, \ldots, m$, to produce a set of queries, $Q = \{Q_1, Q_2, \ldots, Q_m\}$. In particular, for a given remote host H_j, k sub-queries are generated based on the index i_j and the random number r_j, that is,

$$
\begin{array}{l}
Q_1 = \{Q^1{}_i(i_1, r_1) \mid i = 1, 2, \ldots, k;\} \\
Q_2 = \{Q^2{}_i(i_2, r_2) \mid i = 1, 2, \ldots, k;\} \\
\quad\ldots\ldots \\
Q_m = \{Q^m{}_i(i_m, r_m) \mid i = 1, 2, \ldots, k;\}
\end{array}
$$

For a given piece of code C_i, m sub-queries, $\{Q^1{}_i, Q^2{}_i, \ldots, Q^m{}_i\}$ where $i = 1, 2, \ldots, k$, are carried and executed at m remote hosts respectively. All the answers returned from the remote hosts can be represented as follows:

$$
\begin{array}{l}
A_1 = \{A^1{}_i(x^1, Q^1{}_i(i_1, r_1) \mid i = 1, 2, \ldots, k\} \\
A_2 = \{A^2{}_i(x^2, Q^2{}_i(i_2, r_2) \mid i = 1, 2, \ldots, k\} \\
\quad\ldots\ldots \\
A_m = \{A^m{}_i(x^m, Q^m{}_i(i_m, r_m) \mid i = 1, 2, \ldots, k\}
\end{array}
$$

It is important to notice that for a given A_j, there are in fact k candidate answers available. The intended result can be derived from the k answers, provided that at least $k - t$ candidate answers contain the correct results. In other words, our system is designed to tolerate up to t faults and/or attacks. For each of m remote hosts, the intended and final answer can be reconstructed using the following reconstruction functions:

$$
\begin{array}{l}
R_1(i_1, r_1, A^1{}_1(x^1, Q^1{}_1(i_1, r_1)), \ldots, A^1{}_k(x^1, Q^1{}_k(i_1, r_1))) = x^1{}_{i1} \\
R_2(i_2, r_2, A^2{}_1(x^2, Q^2{}_1(i_2, r_2)), \ldots, A^2{}_k(x^2, Q^2{}_k(i_2, r_2))) = x^2{}_{i2} \\
\quad\ldots\ldots \\
R_m(i_m, r_m, A^m{}_1(x^m, Q^m{}_1(i_m, r_m)), \ldots, A^m{}_k(x^m, Q^m{}_k(i_m, r_m))) = x^m{}_{im}
\end{array}
$$

3. The SeCode System Architecture

Figure 1 outlines an architectural framework for our system. Any host that sends mobile code to remote hosts contains four key components (or services). The *request manager* is responsible for generating sub-requests from a user request, and the *mobile code dispatcher* is responsible for producing the corresponding pieces of the code and sending them to remote hosts. The *mobile code collector* will collect the returning objects and results, and the *result manager* will finally reconstruct the desirable results based on all the information and data back from remote hosts, and perform a diagnosis algorithm for identifying faulty hosts and malicious attacks.

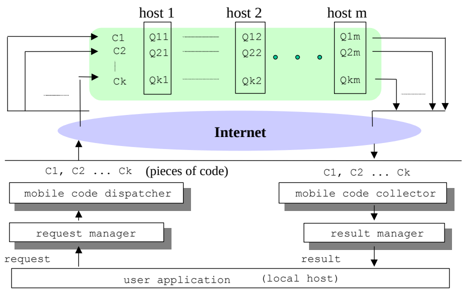

Figure 1. The SeCode System Architecture

We have taken a system approach rather than a programming language approach. Our strategy is to build fault tolerance and security into the system (or platform) and applications themselves, rather than attempt to introduce reliability and security patches afterward. The development of a sound architectural framework is the essential part of our approach. Within this framework, the infrastructure services (e.g. fault tolerance and security) will be provided and incorporated into mobile code platforms so as to facilitate the fault-tolerant and secure execution of mobile code.

Reference

[1] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private Information Retrieval," *J. of the ACM*, 45(6), pp. 965-982, Nov. 1998.

[2] W. Jansen, "Countermeasures for Mobile Agent Security," *Computer Communications*, Elsevier Science BV, Nov. 2000.

[3] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov, "NAP: Practical Fault tolerance for Itinerant Computations," in *Proc. 19th IEEE ICDCS*, Austin, Texas, Jun. 1999.

[4] L. L. Kassab and J. Voas, "Towards Fault-Tolerant Mobile Agents," in *Workshop on Distri. Comput. on the Web*, Rostock, Germany, 1999.

[5] A. Mohindra, A. Purakayastha and P. Thati, "Exploiting Non-determinism for Reliability of Mobile Agent Systems," in *Proc. of 30th IEEE FTCS*, New York, USA, 2000.

[6] G. P. Picco, "Mobile Agents: An Introduction," in the Journal of Microprocessors and Microsystems, (A. Corradi ed)., 25(2), Elsevier Science, Apr. 2001.

[7] L. M. Silva, V. Batista and J. G. Silva, "Fault-Tolerant Execution of Mobile Agents," in *Proc. of DSN'2000*, New York, USA, 2000.

[8] M. Strasser and K. Rothermel, "System Mechanisms for Partial Rollback of Mobile Agent Execution," in *Proc. of ICDCS'2000*, Taipei, Taiwan, 2000.

[9] B. S. Yee, "A Sanctuary for Mobile Agents," in *Secure Internet Programming*, vol. 1603, (J. Vitek and C. Jensen, eds), Springer-Verlag, pp. 261-274, 1999.