available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

What the heck is this application doing? – A security-by-contract architecture for pervasive services☆

N. Dragoni[a], F. Massacci[b], T. Walter[c,*], C. Schaefer[c]

[a]Technical University of Denmark, Richard Petersens Plads, 2800 Lyngby, Denmark
[b]University of Trento, Via Sommarive 14, I-38050 Povo (Trento), Italy
[c]DOCOMO Euro-Labs, Landsberger Strasse 312, 80935 Munich, Germany

ARTICLE INFO

Article history:
Received 15 October 2007
Received in revised form
29 May 2009
Accepted 22 June 2009

Keywords:
Security-by-contracts
Security architecture
Pervasive downloads
Pervasive services
Policies
Policy enforcement

ABSTRACT

Future pervasive environments are characterized by non-fixed architectures made of users and ubiquitous computers. They will be shaped by *pervasive client downloads*, i.e. new (untrusted) applications will be dynamically downloaded to make a better use of the computational power available in the ubiquitous computing environment.

To address the challenges of this paradigm we propose the notion of *security-by-contract* ($S \times C$), as in programming-by-contract, based on the notion of a mobile contract that a pervasive download carries with itself. It describes the relevant security features of the application and the relevant security interactions with its computing environment. The contract can be used to check it against the device policy for compliance.

In this paper we describe the $S \times C$ concepts, the $S \times C$ architecture and implementation and sketch some interaction modalities of the $S \times C$ paradigm.

1. Introduction

Security and trust have been identified as key issues in the pervasive computing vision from their earliest inception (Weiser, 1993). Indeed, the paradigm of pervasive services (Bacon, 2002) envisions a nomadic user traversing a variety of environments and seamlessly and constantly receiving services from other portables, hand-helds, embedded or wearable computers. Bootstrapping and managing security of services in this scenario is a major challenge as downloaded code might be malware using too many resources of the device or even harm the device.

We argue that the challenge is bigger than the "simple" pervasive service vision because it does not consider the possibilities that open up when we realize that the smart phone in our pocket has *already* more computing power than the PC encumbering our desk 15 years ago.

Current pervasive services, including context-aware services, do not exploit the computational power of the mobile device. Information is provided to the mobile user anywhere but the computing infrastructure is centralized (Harter et al., 2002). Even when it is decentralized to increase scalability and performance (Diot and Gautier, 1999; Chakraborty et al., 2007), such distribution does not exploit the device's computing power.

We believe that the future of pervasive services will be shaped by *pervasive client downloads*. When traversing environments the nomadic user does not only invoke services

☆ This work is partly supported by the project EU-IST-STREP-S3MS (www.s3ms.org). A preliminary, much shorter version of this paper has been accepted to IEEE SecPerU-07.

* *Corresponding author.*
E-mail addresses: ndra@imm.dtu.dk (N. Dragoni), massacci@dit.unitn.it (F. Massacci), walter@docomolab-euro.com (T. Walter), schaefer@docomolab-euro.com (C. Schaefer).

0167-4048/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.06.005

according to a web-services-like fashion (either in push or pull mode) but also download *new* applications that are able to exploit the computational power of the user's device. For instance, in order to make a better use of the services available in the environment.

Client downloads create new threats and security risks on top of the "simple" pervasive service invocation because it violates the model of mobile software download behind the Java (Gong and Ellison, 2003) and .NET mobile security architectures (LaMacchia and Lange, 2002; Paul and Evans, 2004):

- Most pervasive software producers are small and medium sized enterprises (SME) which cannot afford the costs of certification necessary to obtain an operator's certification and thus the downloaded application will not run as trusted code.
- A pervasive download is essentially untrusted code whose security properties we cannot check and whose code signature (if any) has no degree of trust.
- According to the classical security model the downloaded untrusted code should be sandboxed, its interaction with the environment and the device's own data should be limited.
- Yet this is against the whole business logic, as we made this pervasive download precisely to have lots of interaction with the pervasive environment!
- In almost all cases this code is trustworthy, i.e. not harming the host system, is being developed to exploit the business opportunities of pervasive services.

1.1. Contributions of the paper

Given the above considerations, our contributions are as follows:

- We develop the concept of *Security-by-Contract* ($S \times C$) as a mechanism to make the trust-less download of code possible. $S \times C$ covers all stages of the software life-cycle: from design and development to execution. The key idea behind $S \times C$ is that the result of each stage of the software life-cycle is verified against defined properties and, if verification is successful, it is forwarded to the next stage. Besides generic hardware and software platform properties, verification may as well take policies into account. Mobile users have an interest that downloaded code respects their policies, e.g., which communication resources can be used to what extent so that malware is prevented from using too many resources. Mobile operators have an interest that the application does not harm the functioning of the mobile device. $S \times C$ enables the trustworthy (modulo above mentioned properties and policies) execution of the downloaded code on the user's mobile device.
- Although designed to cover the software life-cycle, $S \times C$ provides the flexibility to choose among the tools, to skip verifications and to enter the process at any point of the life-cycle. Where to enter the workflow depends on the available data.
- Although the computational power of mobile devices is steadily increasing, it may not be sufficient to perform some of the verification steps on the device itself. To cope with this situation, the $S \times C$ paradigm allows for an outsourcing of some verifications to (trusted) third parties. Involving third parties, however, requires that the communication between the involved parties is being protected. Thus, the $S \times C$ concept has to be embedded into an architecture that provides access to the $S \times C$ services for performing mentioned verification and which is supported by a security service.

1.2. Outline of the paper

We start discussing related work (Section 2) to motivate the need for a generic security framework for pervasive services. Then we describe the Security-by-Contract $S \times C$ paradigm in detail (Section 3) and discuss the phases of the software life-cycle and applicable verification techniques. Further, we discuss our layered security architecture and security services (Section 4) supporting the $S \times C$ paradigm, and discuss the vulnerabilities and mitigation strategies of the employed security services (Section 5). We highlight our implementation of the $S \times C$ architecture (Section 6) and sketch some interaction modalities of the $S \times C$ paradigm (Section 7) before we conclude.

2. Related work

Four main approaches to mobile code security can be broadly identified in the literature: *sandboxes* limit the instructions available for use, *code signing* ensures that code originates from a trusted source, *proof-carrying code (PCC)* carries explicit proof of its safety, and *model-carrying code (MCC)* carries security-relevant behavior of the producer's mobile code.

2.1. Sandbox security model

This is the original security model provided by Java. The essence of the approach (Gong, 1997) is that a computer entrusts local code with full access to vital system resources (such as the file system). It does not, however, trust downloaded remote code (such as applets), which can access only the limited resources provided inside the sandbox. The limitation of this approach is easily recognizable: it can provide security only at the cost of unduly restricting the functionality of mobile code (e.g., the code is not permitted to access any files). The sandbox model has been subsequently extended in Java 2 (Gong and Ellison, 2003), where permissions available for programs from a code source are specified through a security policy. Policies are decided solely by the code consumer without any involvement of the producer. The implementation of security checking is done by means of a runtime *stack inspection* technique (Wallach and Felten, 1998).

In .NET each assembly is associated with some default set of permissions according to the level of trust. However, the application can request additional permissions. These requests are stored in the application's *manifest* and are used