Security-By-Contract (S×C) for Mobile Systems*

or how to download software on your mobile without regretting it.

N. Dragoni F. Massacci K. Naliuka
Dipartimento di Ingegneria e Scienza dell'Informazione
University of Trento, ITALY
surname@disi.unitn.it

Abstract

We present the notion of *Security-by-Contract* (S×C), a mobile contract that an application carries with itself. The key idea of the framework is that a digital signature should not just certify the origin of the code but rather bind together the code with a contract. In a nutshell, a contract describes the security relevant interactions that the mobile application could have with the mobile device. The contract should be accepted by the platform (if compatible with the policy) at deployment time, and its enforcement guaranteed, for instance by in-line monitoring.

A short *live demo* with real phones will be done at the end of the talk.

1 Introduction

Mobile devices are increasingly popular and powerful. Yet, the growth in computing power of nomadic devices has not been supported by a comparable growth in available software.

One of the reasons for this lack of applications is also the security model adopted for mobile phones. The current security model is exemplified by the JAVA MIDP 2.0 [3] and .NET CF [5] approach and is based on *trust relationships*: mobile code is run if its origin is trusted. The level of trust of the "trusted party" determines the privileges of the code by essentially segregating it into appropriate trust domain. The problem with trust relationship, i.e. digital signatures on mobile code, is twofold. At first we can only reject or accept the signature. The second (and major) problem, is that *there is no semantics attached to the signature.*

This is a problem for both code producers and consumers. Mobile code consumers must essentially accept the code "as-is" without the possibility of making informed decisions. One might well trust SuperGame Inc. to provide excellent games and yet might decide to rule out games that keep playing while the battery fells below 20%. At present such choice is not possible. Code producers cannot declare which security actions the code will do: by signing the code they only declare that they did it. The consequence is that injecting an application in the mobile market is a costly operation as SME developers must essentially convince the operators that their code will not do anything harmful. As a result, a lot of code comes at the market uncertified or self-certified. In a word, untrusted.

Such situation has essentially polarized users in two groups: the security paranoids and the technology enthusiasts. From the former perspective:

> `The policy should protect the integrity of the device, and of other applications on the device, from any application that is loaded, i.e. sandboxing. [...] If the agenda data is sensitive, then I NEVER want untrusted applications to access it. This is much simpler than a temporal requirement that an untrusted application cannot have network access if it has looked at sensitive data.`

The life of the technology enthusiasts is riskier but surely better:

> `There is this nice midlet that access my agenda and at 7:50 pops up a window that today is the birthday of these people in the contact list. [...] I'm no longer the only parent who never greets my children's teachers on their birthday. Look, there is also an option to send an SMS with happy birthday to the phone numbers but that would be too expensive with my current subscription.`

*Research partly supported by EU-FP6-IST-STREP-S3MS, and EU-FP7-IP-MASTER.

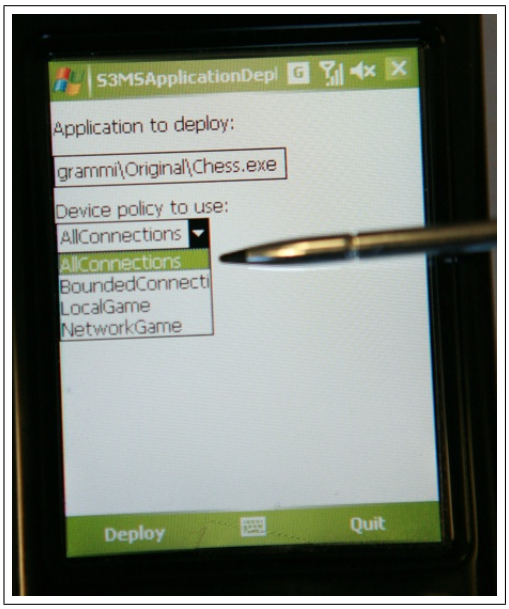

Figure 1: User selects a security policy

Unfortunately, the technology enthusiast cannot just say that access to the agenda is fine provided there is no network connection. If the permission is granted the application may do anything with the obtained information, including sending it to a hackers' web site in Russia. If the required permission is not granted the application becomes completely useless. We need something beyond sandboxing.

2 Security-by-Contract (S×C) as a solution

We present in this paper the notion of *Security-by-Contract* (as in programming-by contract [6]): the digital signature should not just certify the origin of the code but rather bind together the code with a contract [2, 2].

Loosely speaking, a *contract* contains a description of the relevant features of the application and the relevant interactions with its host platform. Among the relevant features, one can list fine-grained resource control (e.g. silently initiate a phone call or send a SMS), memory usage, secure and insecure web connections, user privacy protection, confidentiality of application data, constraints on access from other applications already on the platform.

The mobile code developers are responsible to provide a description of the security behavior that their code provides. By signing the code the developer certifies that the code complies with the stated claims on its security-relevant behavior.

On the other side we can see thaûsers and mobile phone operators are interested that all codes that are deployed on their platform are secure. A mobile platform could specify platform contractual requirements, a *policy*[1], which should be matched by the application's contract.

Figure 1 shows how the user chooses a specific security policy among four pre-set policies.

Example 2.1 *The user might select a* policy *consisting of the following two rules:*

1. *the application can access the PIM but then network connection as disallowed.*
2. *maximum five text messages can be sent by the application.*

If contract represents the security behavior of an application the temptation would be to make such contractual claims arbitrarily complex. Since we argue that contract should be matched by mobile device a complex procedure is likely to defy the very spirit of our proposal.

[1]In the sequel we will refer to policy as the security requirements on the platform side and by contract the security claims made by the mobile code.

1. The application sends no more than a number messages in each session.
2. The application only loads each image from the network once.
3. The delay between two periodic invocations of the MIDlet is at least T.
4. the midlet is suspended if the battery falls below a threshold
5. The application does not initiate calls to international numbers.
6. If the PIM is accessed then no network connection is allowed.
7. The application does not send MMS messages.
8. The application connects only to its origin domain.
9. The application does not use the *FileConnection.delete()* function.
10. The application only receives SMS messages on a specific port.
11. The length of a SMS message sent does not exceed the payload of a single SMS message.

Figure 2: Sample S×C claims for mobile devices

Table 1: Enforcing Security-by-Contract at Different Stages

Development	(I) at design and development time
Deployment	(II) after design but before shipping the application (III) when downloading the application
Execution	(IV) during the execution of the application

Further, a number of independent security requirements analyses for mobile and distributed systems [4, 7, 9] show that detailed contracts are not really necessary. The characteristic feature of these applications is that they need wide access to services to execute correctly. However, the user still wants to control that these services are not abused or misused. Therefore the same permission can be granted or not granted depending, for instance, on previous actions of the midlet or some conditions on application environment.

A suggestive but not indicative list of security policies for mobile devices is give in Figure2

3 Deployment of Contract-Enabled Applications

The overall goal is to give the user the opportunity of downloading an arbitrary program on the device and then to run it safely.

In order to guarantee that an application complies with its desired contract or the policy requested on a particular platform we should consider the stage where such enforcement can be done as shown in Table 1. Enforcing at level (I) can be achieved by appropriate design rules and require developer support; (II) and (III) can be carried out through (automatic) verification techniques. Such verifications can take place before downloading (*static verification* by developers and operators followed by a contract coming with a *trusted signature* and *contract-policy matching*) or as a combination of pre and post-loading operations (e.g., through *in-line monitors* and *proof carrying code*); (IV) can be implemented by *run-time checking*. All methods have different technical and business properties. From an operator's view point:

- working on existing devices would rule out run-time enforcement and favour static analysis, code signing and signature verification on the mobile platform. Monitors may be used (for properties that could not be proved), but on-device proof would then not be possible.
- Operators distrusting the certification process could rely on run-time checks, at the price of upgrading devices' software. Monitors could be used and contracts could be verified on the device itself.
- An operator who wants to be able to run existing applications would prefer run-time enforcement.

In summary this gives us the workflow from Fig.3.

There might be cases in which the claimed contract does not satisfy the policy but the user wants to still download the application. For this reason, in the S×C framework we offer the possibility to rewrite the downloaded program inserting (in-lining) hooks that notify the monitor before and after each security-relevant event. The in-lining is performed directly on the device, so that we do not need to rely on any third party for that. As the result of the in-lining each security-relevant method is executed only if it has been allowed by the policy.

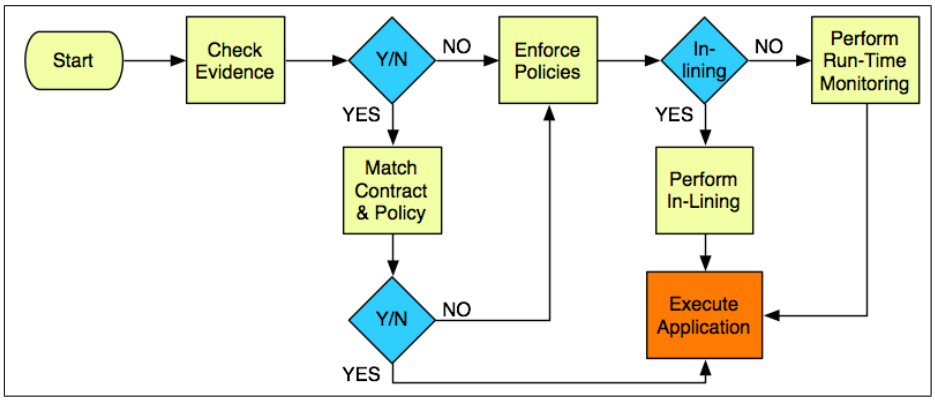

Figure 3: Application/Service Life-Cycle

Scenario. The user designs his security policies or, if he or she is not competent enough, picks the predefined one from the operator. Then the policies are automatically compiled in the form suitable for the runtime monitoring. The compiled policies are deployed at the device. When the application is downloaded and installed at the device it might be in-lined with any of the user policies. After the user selects the policy the application is rewritten and the calls to the monitor are inserted before and after each call to the method mentioned in the policy. Then when the application is run the monitor checks whether its execution satisfies the policy, and is the violation is detected a security exception is raised in the application. If the application does not handle the security exceptions then it terminates. Otherwise if the developers took the security exceptions into consideration the application may be able to proceed (or at least to terminate gracefully, notifying the user about the situation). But in any case the execution that violates the policy will not be possible.

Example 3.1 *Figure 4 shows two HTC P3600 smartphones playing chess by sending each other SMSs. The smartphone on the left has no monitoring framework installed, so the game is allowed to send as many SMSs as it will. The monitoring system of the phone on the right detected that the user-defined limit of the sent messages was exceeded. For this reason the game is terminated through the security exception.*

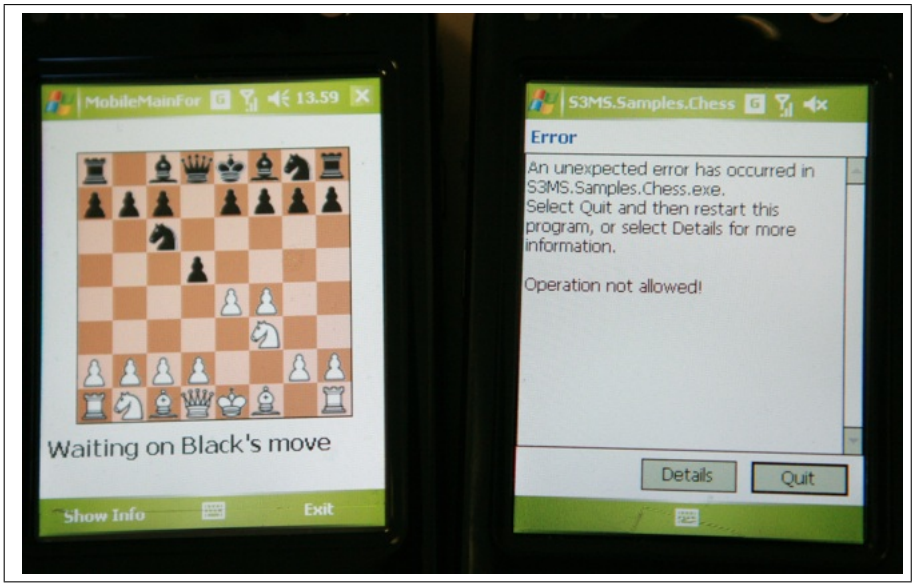

Figure 4: Policy violation and error message

4 Conclusion

In this paper we have presented the Security-by-Contract (S×C) framework, which aims to build the basis for the opening of the software market of nomadic devices (from smart phones to PDA). The key intuition is the concept of *security contrac*t, a mobile contract that an application carries with itself. The rationale of the approach is that a digital signature should not just certify the origin of the code but rather bind together the code with a contract. We argue that Security-by-Contract would provide a semantics for digital signatures on mobile code thus being a step in the transition from trusted code to trustworthy code. S×C will not replace, but enhance today's security mechanisms, and will provide a flexible, simple and scalable security mechanism for future mobile systems

References

[1] L. Desmet et al. Security-by-contract on the .NET platform. *Information Security Technical Report.* Elsevier 2008.

[2] N. Dragoni, et al. Security-by-Contract (SxC) for Software and Services of Mobile Systems. In *At Your Service, Selected Papers on research on Software and Services.* MIT Press 2008.

[3] L. Gong and G. Ellison. *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation.* Pearson Education, 2003.

[4] M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. Usage control requirements in mobile and ubiquitous computing applications. In *Proc. of the Int. Conf. on Sys. and Net. Comm. (ICSNC 2006)*, page 27. IEEE Press, 2006.

[5] B. LaMacchia and S. Lange. *.NET Framework security.* Addison Wesley, 2002.

[6] B. Meyer. Building bug-free O-O software: An introduction to Design by Contract(TM). Availabe at `http://archive.eiffel.com/doc/manuals/technology/contract/`.

[7] MOBIUS Project Team. Framework- and application-specific security requirements. Public Deliverable of EU Research Project D1.2, MOBIUS - Mobility, Ubiquity and Security, Report available at http://mobius.inria.fr, 2006.

[8] R. Sekar, V.N. Venkatakrishnan, S. Basu, S. Bhatkar, and D.C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP-03)*, pages 15–28. ACM Press, 2003.

[9] A. Zobel, et al. Business case and security requirements. Public Deliverable D5.1.1, EU Project S3MS, Report available at www.s3ms.org, October 2006.