Secure Software Installation in a Mobile Environment

Andreas P. Heiner
Nokia Research Center
Itamerenkatu 11-13
Helsinki, Finland
Andreas.heiner@nokia.com

N. Asokan
Nokia Research Center
Itamerenkatu 11-13
Helsinki, Finland
N.Asokan@nokia.com

ABSTRACT

Software security in mobile devices today is done by granting privileges to software, usually based on code signing. The cost of obtaining signatures and meeting strict quality requirements deters hobbyist developers from participating and contributing to application development. If a certain piece of software does not come with an acceptable signature, the mobile device may give the user the option of deciding whether that software should be granted the requested privileges. Naturally, designing the user interaction for this step without hampering usability and security is tricky. When users are simply prompted whether they want to grant certain privileges to some software, they often do not have enough information to understand the implications of this action.

We propose that using community feedback can be an effective way of helping the user to decide whether to grant privileges to software. Community feedback includes opinions and ratings on both security and functionality attributes of software. We argue that users will use community feedback to decide whether they want to *use* a piece of software and that the decisions to download, install, and grant necessary privileges are implied by the decision to use.

1. INTRODUCTION

Mobile phones used to be closed devices with fixed functionality. In the last decade, they have become increasingly open. Platforms like Symbian OS, Java2 Micro Edition (J2ME), and Windows Mobile allow the possibility to install new applications to extend the functionality of mobile phones. Openness brings in the risk of malware just as in the world of personal computers.

To deal with this risk, designers of mobile device platforms have introduced new security architectures. For example, Symbian OS supports a platform security model, and J2ME has Java security architecture. In all of these architectures, the access control decision to assign privileges to software processes is based either on code signing or on explicit user approval, or a combination thereof. In this paper, we point out the difficulties of the privilege assignment step, and discuss how it can be improved.

Symposium On Usable Privacy and Security (SOUPS) 2007, July 18-20, 2007, Pittsburgh, PA, USA

1.1 Software security models

In Symbian platform security and J2ME security architecture, the software package includes an explicit statement of the privileges needed for the software to do its job. The platform security software on the mobile device will decide whether the requested privileges may be granted. Both security architectures implement code signing to enable a trace-back to a trusted entity that endorses the software.

In Symbian OS platform security, privileges granting is done by the software installer. In case software is signed by an (ultimately) high-level Certificate Authority installation proceeds without further questions. If the signature is not present the user has to explicitly grant these privileges (Figure 1a). The text explaining the warning does not provide relevant information to the user. The "Additional Details" link leads to four pages explaining the general philosophy behind trusted applications; the following four pages list for what resources permissions can be set.

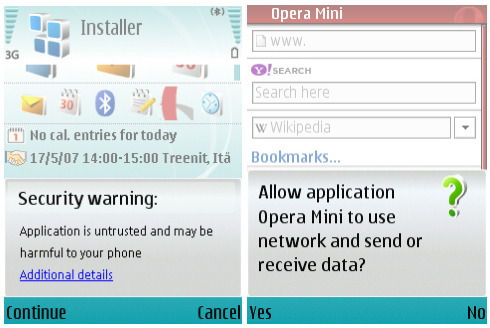

Figure 1 Pop-up screens when a) installing software on a Symbian device and b) run-time privilege requests in J2ME applications.

In contrast to Symbian platform security, in J2ME, privilege granting can take place at the time of access to a resource (Figure 1b). The prompt suggests asking permission to connecting to that URL. The actual function is to open a network connection and send and receive data. The purpose of the prompt is mostly cost control, and to a lesser extent security.

Thus, the two alternative approaches to grant privileges to software have serious drawbacks. Code signing leads to better usability but is expensive and hence discourages out a large portion of the developer community. User prompting avoids the developer cost, but suffers from not being able to communicate the nature of the prompt effectively to the user, and hence could easily lead to a user granting privileges to malicious software.

2. TOWARDS A FRAMEWORK FOR SECURE SOFTWARE INSTALLATION

2.1 Design considerations

Software installation is preceded by the user loosely defining the required functionalities, searching for potential matches, and finally downloading and installing the selected application. The first two steps, definition and search, is a recursive process and very problematic in a mobile environment due to device limitations. Bandwidth may be costly and/or small, and the display of the top-of-the-line phone models is typically 6 times smaller than a low-end laptop. Keyboard input is clumsy at best, and battery resources are also limited. Especially WLAN access is resource demanding.

Obtaining all essential information (functionality, security advisory, privacy advisory) in as few steps is essential. It is equally important that the user keeps control over the process, and makes the final decision [2]. The handset limitations require a concise representation of that information. "Security by admonition" [3] and "Trust in a socio-technical sense" [1] are therefore important elements of the framework. We make the following design considerations

1. Download and installation must be transparent and integral part of the software functionality definition–usage process. Software selection implies usage, and user interaction is only needed when the software is actually used.
2. The installation process is independent of the access technology (Bluetooth, cellular…). From a user perspective the difference is cost rather than technology
3. Access control and installation decisions must be done on the device. Doing so captures push-mode installations of unknown sources
4. Software is certified implicitly by usage patterns. The "community certification" is a community effort of reviewing the software, and reporting suspect software and sites to experts for obtaining privacy- and safety advisories.
5. Users can add descriptive data to the software.
6. Attributes of already installed software can be updated. Safety attributes may change urging the users to update the software, or even de-install the software.
7. All software is safe until found otherwise.

The term "community certification" (consideration 4) deserves some elaboration. One definition of product (software) certification is "established suitability for a specified purpose"[5]. SymbianSigned [4] formulates a set of technical requirements; if all tests are passed the software is Symbian certified (meets the architectural requirements). The usable security research community uses similar phrases as necessary condition for secure software [1][3]. The purpose is defined by the author; the community (having similar semantics in that domain) evaluates the suitability for that purpose by ratings and written reviews.

Consideration 7 is the only pragmatic approach for creating an open and affordable development environment. In fact, this approach is also taken by anti-virus software vendors; they react on reports of ill-behaving software.

The above considerations mitigate most of the usability problems at installation time. Security at runtime is more difficult. One can image a scenario in which the application is used for a given use case. The privileges used by the community are automatically downloaded. Based on the description of the use case by the community the user automatically downloads these settings. The same could apply for access to web sites. However, in all cases the user makes the final decision. The needed information (security advisory, community certification, etc.) is given in a short description (e.g., represented by an icon) on one page, with links to the more detailed information (reason for the warning, who gave the review, etc.).

2.2 Architecture

The high-level architecture envisions an environment in which all applications have a unique identifier that can be calculated from the executable application code. Moreover, each application has a number of searchable metadata (e.g., type, author, keywords, security advisory, etc.). The user can specify the needed functionality, and a search is performed. The metadata of potential applications is downloaded to the mobile device and presented to the user. After some iteration the user selects an application. The application is downloaded, and an integrity check is performed. After that the application is installed without further prompting. The process includes an on-device and in-proximity search. In that case security- and safety advisories may be updated if so desired.

The actual implementation is likely to include a portal where all data are collected. It is also facilitates the search process. However, the final decision to install is made on the handset, after the implicit download via one of the wireless interfaces. Only if the footprint of the application does not match that of the application identifier installation is aborted.

3. STATUS AND FURTHER WORK

At present we are implementing a first version of the proposed architecture. We plan to perform small-scale user studies to verify our assumptions and improve the user interface.

4. REFERENCES

[1] Flechais, I.; Riegelsberger, J. and Sasse, M.A. Divide and Conquer: the role of Trust and Assurance in the design of secure Socio-Technical Systems. In *Proceedings of the 2005 workshop on new security paradigms* (Lake Arrowhead, California, USA September 20-23, 2005)

[2] Stoll, J. and Park, F. Exploring Explicit Security Actions. In *Proceedings of the SIGCHI conference on Human factors in computing systems(CHI '2007)* (San Jose, California, USA, April 28-May 3, 2007)

[3] Yee, Ka-Ping. Guidelines and Strategies for Secure Interaction Design In *Security and Usability: Designing Secure Systems that People Can Use. Cranor, L.F. and Garfinkel, S (eds.)* O'Reilly & Associates 2005 247-274

[4] SymbianSigned.com: test criteria (https://www.symbiansigned.com/app/page/overview/testcriteria)

[5] Wikipedia: certification (http://en.wikipedia.org/wiki/Product_certification