A Secure Smartphone Applications Roll-out Scheme

Alexios Mylonas, Bill Tsoumas, Stelios Dritsas, and Dimitris Gritzalis

Information Security and Critical Infrastructure Protection Research Laboratory
Dept. of Informatics, Athens University of Economics & Business (AUEB)
76 Patission Ave., GR-10434, Athens, Greece
{amylonas,bts,sdritsas,dgrit}@aueb.gr

Abstract. The adoption of smartphones, devices transforming from simple communication devices to smart and multipurpose devices, is constantly increasing. Amongst the main reasons for their vast pervasiveness are their small size, their enhanced functionality, as well as their ability to host many useful and attractive applications. Furthermore, recent studies estimate that application installation in smartphones acquired from official application repositories, such as the Apple Store, will continue to increase. In this context, the official application repositories might become attractive to attackers trying to distribute malware via these repositories. The paper examines the security inefficiencies related to application distribution via application repositories. Our contribution focuses on surveying the application management procedures enforced during application distribution in the popular smartphone platforms (i.e. Android, Black-Berry, Apple iOS, Symbian, Windows Phone), as well as on proposing a scheme for an application management system suited for secure application distribution via application repositories.

Keywords: Smartphone, Security, Mobile Applications, Software Roll-out.

1 Introduction

Smartphones appear to be devices that fall really close to enhance Weiser's vision of ubiquitous computing [1]. Their small size, together with their mobility and connectivity capabilities, as well as their multi-purpose use, are some of the reasons for their vast pervasiveness over the last few years [2].

Malicious software, or malware [3], [4], [5] has also appeared [6] in smartphones, but its occurrence and severity currently appear to be limited. Nonetheless, recent reports show that the risk introduced by malware on smartphones is severe and contingent [7], [8]. In addition, by considering that smartphones extend the infrastructure perimeter of an organization, the impact and risk of the threats introduced by mobile devices, and especially from malware attacks, is expected to be amplified [9].

Apart from the increasing smartphone sales [2], the annual downloads of smartphone applications that are distributed from application repositories are expected to be increased by 117% in 2011 [10]. Moreover, several popular web applications (e.g. YouTube) and social networks (e.g. Facebook) are being accessed on mobile devices through native applications, instead of their web browser interface. In this context, smartphones contain a vast amount of users' personal data and, thus, introduce a serious privacy threat vector [11], [12], [13]. These data are augmented with smartphone sensor data (e.g. GPS) and data created by everyday use (personal or business),

S. Furnell, C. Lambrinoudakis, and G. Pernul (Eds.): TrustBus 2011, LNCS 6863, pp. 49–61, 2011.

making the device a great source of data related to the smartphone owner. This data source is attractive to attackers trying to harvest data to increase their revenues (e.g. with blackmail, phishing, surveillance, espionage attacks, etc.). In addition, everyday use of smartphones by non-technical and non-security savvy people increases further the likelihood of using smartphones as a security and privacy attack vector.

Currently, every smartphone platform applies different and non-standardized application submission procedures, while their effectiveness is controversial [14]. The security scheme [15] of smartphone platforms should, under these circumstances, be extended to provide a managed application repository, where specific security controls are enabled towards the protection from malware spreading in the repository.

This paper examines security inefficiencies related to application distribution via application repositories. Our contribution focuses on surveying the application management procedures enforced by current smartphone official repositories, as well as on proposing a scheme for an application management system suited for application submission in the official application repositories.

The rest of the paper is organized as follows: Section 2 provides background information, regarding current smartphone operating systems; in Section 3 existing application management approaches are presented. The suggested scheme for an application repository system is presented in Section 4, followed by discussion in Section 5. The paper concludes in Section 6.

2 Smartphone Platforms

In this section the security schemes of the most popular smartphone platforms, namely Android, BlackBerry, iOS, Symbian, and Windows Phone are summarized. Security mechanisms employed for physical device protection (e.g. data encryption, anti-theft solutions, etc.) are not analyzed herein.

2.1 Android Platform

Android OS is a Linux-based open source operating system maintained by Google. Core elements of the Android security scheme [16] are the application permissions that control access to protected resources. By default, every application runs in a sandboxed environment and requests permission authorization to be granted by the user at installation time. No further permission checks are made during application execution.

A developer distributes her application either in the official application repository maintained by Google, the Android Market, or in other sources (e.g. Amazon Appstore for Android). Android does not enforce any restriction in the installation of applications originating outside its repository. Nonetheless, Google developed technologies to remove applications [17] from devices and the Android Market in case they pose a threat to the Android platform. Moreover, applications in the Android Market are provided to end-users without being previously tested for malicious behavior. Hence, a developer must only provide her Google account credentials and pay a small fee for application distribution in the Android application repository. It is evident that in this context, potential malicious developers can use the Android Market as a malware distribution point.

The Android security scheme requires every application to be digitally signed by its developer. Nonetheless, the developer's certificate does not mandatorily need to be

signed by a trusted certificate authority. Thus, applications are digitally signed with self-signed certificates, providing only poor source origin and integrity protection. This preserves the anonymity of a potential attacker, since the certificate is not verified by a Trusted Third Party (TTP).

2.2 BlackBerry Platform

The BlackBerry OS is an operating system maintained by Research In Motion Inc. (RIM). The BlackBerry security scheme [18] enforces restrictions to third party application access to protected APIs by mandating application signing with a cryptographic key provided by RIM [19]. A developer must pay a (small) fee to obtain a valid RIM key pair. However, since this process does not include any application testing, it only provides poor source origin and code integrity, without offering any assurance about third party application validity and/or security level.

Similar to the Android platform, a developer distributes her applications either in the official application repository, the BlackBerry App World, or outside the official repository. Application distribution in the official repository requires registration for a vendor account. However, before application publication in the repository, the application is not examined for malicious behaviour by RIM. Moreover, the employment of a remote application removal mechanism is not documented by the platform's security scheme, but RIM can potentially restrict execution by revoking the key pair of a particular developer.

2.3 Symbian Platform

Symbian OS is an operating system maintained by Nokia. The cornerstone in Symbian's security scheme [20] is the use of capabilities for defining restrictions to sensitive platform APIs. Basic functionality (e.g. network access) is granted during application installation by the user, whereas access to more sensitive APIs is only granted by device manufacturers and after the application's certification by Symbian Signed [21]. Application signing is mandated for application installation. The signing process ensures that the application is not using API, apart from the ones corresponding to the application's signing level. If the application uses only basic API, the developer can self sign it [22]. The smartphone user will be prompted with security warnings at installation time, since the signing key is not trusted. To eliminate the warnings and access sensitive capabilities the developer submits her application to Symbian Signed.

Applications are not required to reside in the official application repository, the OVI store, to be installed in Symbian devices. For application submission in the official repository a developer must [23]: (a). register as an OVI publisher, (b). pay a one-time registration fee and (c). submit an application that complies with the Symbian Signed Test Criteria [21]. It must be noted that the criteria include application scanning for malicious code presence.

2.4 iOS Platform

iOS is an operating system maintained by Apple and executed in Apple smartphones and tablets (i.e. iPhones, iPADs). iOS security scheme only permits the installation of applications that have been signed by Apple [24] and are available in the official application repository, the App Store. Before being signed, an application is tested for

its functionality consistency and malicious behaviour. However, the testing process and criteria applied by Apple are not publicly available. For application submission in the App Store, the developer incurs an annual enrollment cost.

Once an application is installed on a device it runs on a sandboxed environment, but the user neither controls, nor is prompted when the application accesses some OS' sensitive resources. All the device resources that are available to applications in iOS version 3 are presented in [25].

2.5 Windows Phone Platform

Windows Phone is an operating system maintained by Microsoft. The security scheme of Windows Phone [26] is based on the least privilege concept and application sandboxing in conceptual chambers, where access to protected resources is granted via capabilities. Third party applications are executed in a least privileged chamber, where access to resources is controlled by capabilities that are granted by the user at installation time and cannot be elevated during execution time.

The Windows Phone security scheme permits third party application installation only from the official application repository, the App Hub. Before application submission in the App Hub the application is tested and the developer is authenticated during registration, in an attempt to maintain a managed application repository. During registration a developer pays an annual registration fee. According to [27], developer authentication is applied to hinder unauthorized developers from using a company's brand name and to assure the users that applications are authentic and their sources are known. Each submitted application is tested for compliance with Windows Phone Application Certification Requirements [28]. The requirements apart from testing the application's functionality and performance involve security tests for malware detection. Moreover, Microsoft employs a remote application removal mechanism to remove malicious applications that manage to enter the application repository and Windows Phone devices.

3 Current Application Management Approaches

From the above-mentioned smartphone security schemes, it is obvious that a reliable security scheme must include an *Application Management System (AMS)* providing a managed application repository. The AMS must impede malicious applications from entering the repository and be able to authenticate their developers. Therefore, the AMS must include secure and robust procedures for developer registration and application submission. In this context an AMS must at least include mechanisms that provide: (a). Application Integrity, (b). Application Testing, (c). Remote Application Removal, (d). Application Testing Documentation, and (e). Developer Strong Authentication. These mechanisms are described and analyzed in the following paragraphs.

Application Integrity ensures that an application's binary is not altered, e.g. by malicious code injection in pirated versions of the application. As mentioned previously, in all smartphone platforms - apart from the Android platform - the security scheme mandates application digital signing with a certificate controlled by the platform. On the contrary, Android allows users to sign applications with custom self-signed certificates not validated by a TTP. As a result, a malicious developer may download and

repack an application with a new certificate and submit it to the Android Market or in an alternative application repository. Apart from monetary loss to the original application's developer, a rogue developer can infect the application with malware, compromising the security of Android devices [14].

Application Testing employs static and/or dynamic binary analysis to ensure application functionality reliability, official API usage and rational resource consumption. It typically contains tests for copyright infringements and, in some platforms, security testing [21], [28]. In a managed application repository, we argue that the security testing process should be mandatory. This will ensure that malware cannot easily be spread through the application repository. Only Symbian, iOS and Windows Phone platforms mandate application testing before submission in their repositories, whereas it is unclear if Apple's iOS employs security testing procedures.

Remote Application Removal also referred as "application remote kill switch" ensures that a malicious application will stop being executed in smartphone devices, if it has not been detected during application testing. The security scheme must ensure that (a) the mechanism will not be used for application censorship and (b) that it is conformant with legislation protecting access to a user's device. Among the surveyed smartphone platforms, only Symbian and BlackBerry do not use a documented remote application removal mechanism.

Application Testing Documentation on the one hand mandates developers into submitting applications that satisfy strict requirements and, on the other hand, informs smartphone users about these testing criteria before application acceptance. From the surveyed platforms only Symbian and Windows Phone document application tests.

Developer Strong Authentication prevents unauthorised developers from using a company's brand name and assures the application repository users that the applications are authentic and their sources are known. All smartphone platforms examined in this paper do not enforce strong authentication during developer registration. The platforms require a fee paid via a credit card during registration, but this does not imply reliable validation, since attackers may use credit cards acquired from the underground market. Windows Phone is the only platform trying to verify the developer identity [27] by outsourcing identity verification to GeoTrust. However, an attacker can use an ID acquired in the underground market (identity theft) or use a fake ID service, e.g. the service provided in [29] during her registration [27], [30], [31].

Table 1. Current Application Management Approaches

Management Functionality	Android OS	BlackBerry OS	Symbian OS	iOS	Windows Phone
Application Integrity	×	✓	✓	✓	✓
Application Testing	×	×	✓	✓	✓
Remote Application Removal	✓	×	×	✓	✓
Application Testing Documentation	×	×	✓	×	✓
Developer Strong Authentication	×	×	×	×	×

The adoption of the management functionalities by the surveyed security schemes is summarized in Table 1 above.

4 Application Repository System Scheme

In this section an Application Repository System Scheme (ARSS) providing the entities and AMS procedures required for a managed application repository is proposed. The proposed ARSS consists of four main entities, namely: Developer, User, Application and Application Repository. A detailed analysis of the entities, as well as, their role in our scheme is described in the following sections. The proposed scheme satisfies the requirements presented in the previous section, regarding an efficient and robust application management system. Our scheme is also cross-platform, since its definition is not dependent on any smartphone platform. Finally, its underlying security mechanisms are fully documented and extensible.

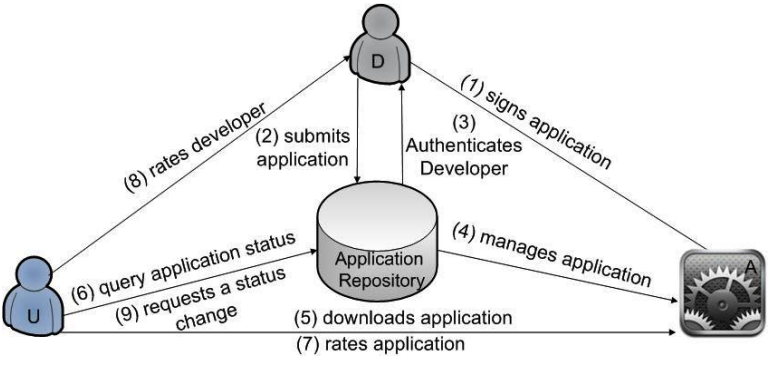

Fig. 1. Entities in the Application Repository System Scheme (the "pyramid")

An overview of the scheme is given in Fig.1. Initially, a developer signs the application and submits it in the application repository. The application repository AMS (referred to as AMS) authenticates the developer and tests the application. If the application is accepted for submission, then its lifecycle in the repository starts. During this lifetime, users are able to download the application, query for an application status, request a status change and optionally rate the application and/or the developer.

4.1 The Developer Entity

In the proposed scheme, the developer - upon application implementation - hashes the application binary. In the sequel, the hash is signed and inserted in the application file, ensuring the application's binary integrity. For the signing process the developer creates a unique signing key contained in a digital certificate. This certificate (referred hereinafter as *application certificate*) contains a subset of the developer's identity data, e.g. her logo, her employer brand and application data, e.g. application title, version, web links etc. The decision to allow developers to create *application certificates* provides flexibility to the developers and the application repository owner. The

former avoids certificate creation costs from a TTP and delays stemming from this creation procedure. The latter avoids computation costs stemming from the creation, management and delivery of each *application certificate*.

In the sequel, the developer includes identity information in the application binary by signing the *application certificate* with a certificate verified by a Public Key Infrastructure (PKI) TTP (referred hereinafter as *developer certificate*). As a result, the certificate chain shown in Fig. 2 is created. In this paper, we propose the use of qualified certificates [32] as *developer* certificates, which apart from providing strong developer authentication, they also provide legal equivalence of a hand-written signature. In this context, the AMS could mandate developers to digitally sign a Computer Misuse Act (CMA) compliant [33] or an equivalent statement. Each developer could state in this statement that she is providing an application which is not impairing the repository management system functionality or the end user device. Nonetheless, the use of qualified certificates and the legal binding of the developers is not mandatory in this scheme. However, the implementation of the proposed scheme must use developer certificates providing strong developer authentication.

Fig. 2. Certificate Chain in the ARSS

Subsequently, the developer concatenates the signed hash and inserts the developer and application certificates in the application. Then, if the developer submits an application for the first time, she enrols to the application repository. The developer provides authentication data and pays a registration fee during enrolment.

4.2 Application Repository Entity

The application repository entity is responsible for developer authentication and application management in the ARSS - i.e. controlling application (a). submission, (b). testing, and (c). remote removal - during the application's lifecycle in the repository. In addition, the AMS uses mechanisms to deter attackers from inserting malicious applications in the repository and misusing its operations and resources.

Strong developer authentication is required during application submission to: (a). avoid application spoofing\phishing attacks, (b). bind the developer with legal responsibilities (e.g. conformance with CMA – like legislation), and (c). give penalties to miscreant developers. In this paper, qualified certificates are proposed for user authentication and hence user management (e.g. user identity checking, secure storage of personal data, etc.) is outsourced to a PKI TTP.

The developer uses her *developer certificate* during developer enrolment. The validity of the certificate is verified and part of its contents are parsed and stored in the application repository. The repository examines only the developer's certificate validity in subsequent application submissions. If the certificate is invalid, then the application submission is rejected and the developer is informed via email for the rejection reasons. The enrolment in the application repository ends with the reception of an

enrolment fee, via a credit card linked to a valid bank account. The credit card number is securely stored, in case monetary penalties are given to miscreant developers. The registration fee, as well as, the penalty monetary amounts are system parameters and must be carefully selected so as: (a). to deter attackers from enrolling and misusing the repository's resources, and (b). not to deter freelancer developers from enrolling to the application repository.

In the sequel, the application hash is verified, ensuring the application's integrity. Similarly, if the integrity check fails the developer is informed via an email. Upon hash verification the application undergoes application testing where the conformance to testing criteria, such as [21], [28] is verified. These criteria must control many application execution aspects and may involve manual inspection or automated testing via static or dynamic binary analysis.

Although this paper does not focus on the application testing procedure, we argue that an application upon submission in the repository, must be tested against a variety of criteria such as: (a). successful execution start and termination, (b). protected API use, (c). unofficial API avoidance, (d). malicious code presence, (e). device resource consumption, (f). application's graphical interface consistency, (g). brand misuse - copyright abuse, etc. Nonetheless, the testing criteria and procedures that the AMS implements must be documented. This informs expert users about the tests and ensures that application censorship is avoided in the repository.

If the application conforms to the testing criteria, test metadata are appended in the application. The AMS may mandate a specific structure and content in the application metadata to facilitate the application testing procedure. Subsequently, the AMS hashes and signs the application. For application signing the AMS uses a certificate that is verified by a PKI TTP. Signing and hashing the application by the ARSS provides, apart from application integrity, protection against malicious ARSS spreading malware through rogue application repositories.

The AMS removes applications from the repository in case they are later proved to be malware. Upon application removal, the AMS has two options: (1). removing a class of applications implemented by the same developer, or (2). removing an individual application from the application repository. In the first case, the AMS inserts the developer's certificate in a blacklist. Similarly, in the second case, the AMS places the application's certificate in a blacklist. In both cases, the AMS gives the miscreant developer monetary penalties. Optionally, the AMS may take legislation action against the developer, especially when the developer has signed a CMA-conformant, or an equivalent, statement with a qualified certificate. In addition, an ARSS user is able to request an application status change, if she can prove that it contains malware. In this scenario the AMS re-checks the application and if malware is found, the application is removed. The user must pay a fee for her request, where the fee is a system parameter, to avoid application removal service misuse by attackers. If the application is found to be malicious then the user is refunded and given a reward (e.g. application discount). This would motivate users to report malicious applications.

4.3 The Application Entity

A conceptual application structure in the ARSS is depicted in Fig. 3. As discussed before, the developer hashes, signs, and inserts the developer hash in the application

file, providing binary integrity. In this paper the use of a hashing algorithm belonging to the SHA-2 family [34] is recommended, since it is widely implemented and resistant to collisions [35]. Furthermore, the application contains a section for metadata, containing data required for application execution, or providing information aiding the repository's application management system, e.g. requested permissions, imported libraries, developer and application certificates, etc.

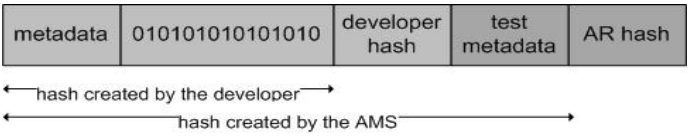

Fig. 3. Application structure in the ARSS

The application contains test metadata that depend on the application testing mechanism that the ARSS employs. At minimum, they must include a vector indicating the execution or not of a test, the test date, etc. These metadata must not be space demanding. Towards this direction, the test metadata can be stored in an online database maintained by the ARSS, and only the AR hash may be included in the application file along with a link to the online database entry. This decision depends on implementation details and our scheme is not imposing restrictions about it. Finally, the application structure contains the AMS-signed hash that provides protection against malicious ARSS spreading malware through rogue application repositories.

4.4 The User Entity

The user enrols to the ARSS by providing identity information - where the identity verification is a system parameter - and a valid credit card. The user selects and downloads an application from the application repository. Consequently, the validity of the application's ARSS certificate is examined. During this validity check, we assume that the device has an up-to-date list of known ARSS certificates. If the application is signed with a valid certificate, then the user trusts that the application is managed by an application repository, i.e.: (a). the application's developer has been authenticated, (b). the application repository has used reasonable care and skill to test the application, (c). the application has not been found to contain malware, and (d). the user will be informed in case the application is later proved to contain malicious code.

Subsequently, the user optionally inspects the developer information contained in the application file, as a second level of defence against spoofing attacks, and installs the application. Upon application installation, we assume that the smartphone's security scheme either checks the developer metadata, or the application is installed, since it has been inspected by the application repository. Furthermore, we assume that the security scheme allows any user to manually inspect both the developer and AR metadata stored in the application. The user must be able to decide whether some application permissions will be denied during execution. In this context, the user's control of the application execution in the device is preserved, while he is aware of the security tests that the application successfully passed.

Software in the user's device periodically queries the repository for blacklisted applications. Hence, in this scheme it is assumed that the device connects regularly to the Internet and that this software - which is included in the application's security scheme - queries for application status change upon application launch. Before querying the AMS, the software may occasionally prompt for Internet access authorization, giving the user control on the query frequency. The query frequency is a system parameter and ensures that the device will be updated in predefined intervals and that the AMS will not be overwhelmed with queries.

The application binary is not deleted from the device when an application certificate is blacklisted. Only the user has the authorization to remove the application. This option is given to avoid unfair application censorship in the application repository and to conform the mechanism with legislation protecting access to a user's device.

Furthermore, in the current scheme, the device's security scheme must block the application execution when its certificates are blacklisted. Nonetheless, the user should have authorization to run the application. In this case before its execution the security scheme must: (a). prompt the user with security warnings, (b). revoke all application's permissions, and (c) enable the user to manually inspect and select, which blocked application permissions will be granted again. The ARSS user optionally rates an application / developer w.r.t. application functionality and user friendliness.

5 Discussion

The proposed scheme includes mechanisms that satisfy the security requirements which were not provided by the surveyed smartphone security schemes, namely: (a). application integrity, (b). application testing, (c). remote application removal, (d). application testing documentation, and (e). developer strong authentication. The scheme is cross-platform, since its definition is not dependent on any smartphone platform implementation details. Moreover, its security mechanisms must be fully documented and extensible. The scheme deters malicious developers from submitting malware in the repository, as well as users from misusing the repository's resources, by giving penalties to miscreant activities. It may optionally deter malicious developers from entering the application repository, by using qualified certificates and imposing the digital signing of a CMA, or an equivalent, statement during developer registration.

The proposed scheme does not focus on the application testing criteria. It considers application testing as a black box containing the state of the art of application testing such as [21], [28]. Nonetheless, the application testing mechanisms and criteria in the scheme's implementation must be documented and carefully selected to avoid performance bottlenecks in this component. Furthermore, the proposed scheme is based on PKI certificates verified by TTP providing strong authentication. The scheme employs authentication only during developer registration, to avoid delays in user registration that could deter user access to the ARSS. The developer incurs a certificate creation cost, for the acquisition of a developer certificate, which is equal to the developer registration cost in current official application repositories. Hence, the enrolment cost must be carefully selected in the scheme implementation so as to not deter developers from enrolling to the repository.

6 Conclusions

This paper examined the necessity of an Application Management System (AMS) embedded in the security scheme of smartphone platforms, providing a managed application repository that hinders malicious application submission in the repository, and is able to authenticate their developers. The proposed approach contributes towards this direction by: (a) surveying the application management procedures enforced in current smartphone official repositories, and (b) proposing a scheme for AMS suited for application submission in application repositories.

The proposed scheme is extensible, cross platform and fully documented. It also includes mechanisms that provide: (a). application integrity, (b). application testing, (c). remote application removal, (d). application testing documentation, and (e). developer strong authentication, which are not provided in whole by current smartphone security schemes. To the best of our knowledge, this is the first time that an AMS is proposed to be included in smartphone security schemes.

Future work will focus on the implementation of the scheme in test and real environment, in order to evaluate its effectiveness and address performance issues. In addition, we plan to survey and amend the state of the art application testing procedures of the scheme. Also, we will explore legal issues protecting application repositories from malware submission and alternative developer authentication schemes that provide strong developer authentication at low cost. Finally, we plan to examine security economics of the proposed scheme to determine optimum values for the scheme's monetary parameters (e.g. registration cost, misuse penalties, rewards, etc).

Acknowledgments. This work has been partially funded by the European Union (European Social Fund) and Greek national funds through the Operational Program *Education and Lifelong Learning* of the National Strategic Reference Framework - Research Funding Program: HERACLEITUS II - Investing in Knowledge Society. The work was, also, supported by the SFINX (09SYN-72-419) project, which is partly funded by the Hellenic General Secretariat for Research & Technology, under the Synergasia Programme.

References

1. Weiser, M.: The computer for the 21st century. Scientific American 265(3), 94–104 (1991)
2. Gartner: Gartner Newsroom (accessed April 15, 2011), http://www.gartner.com/it/page.jsp?id=1543014
3. Adleman, L.: An abstract theory of computer viruses. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 354–374. Springer, Heidelberg (1990)
4. Cohen, F.: Computational aspects of computer viruses. Computers & Security 8(4), 325–344 (1989)
5. Kephart, J., White, S.: Directed graph epidemiological models of computer viruses. In: Lunt, T., et al. (eds.) Proc. of IEEE Symposium on Research in Security and Privacy (SP), pp. 343–359. IEEE Press, USA (1991)
6. Hypponen, M.: Malware goes mobile. Scientific American 295(5), 70–77 (2006)
7. McAfee Labs, 2011 Threats Predictions, Technical Report (December 2010)

8. Cisco: Cisco 2010 Annual Security Report (accessed April 15, 2011), http://www.cisco.com/en/US/prod/vpndevc/annual_security_report.html
9. Forrester: Forrester Research (accessed April 15, 2011), http://www.forrester.com/rb/Research/security_of_b2b_enabling_unbounded_enterprise/q/id/56670/t/2
10. Gartner: Gartner Newsroom (accessed April 15, 2011), http://www.gartner.com/it/page.jsp?id=1529214
11. PAMPAS, Pioneering Advanced Mobile Privacy and Security (accessed April 15, 2011), http://www.pampas.eu.org/
12. Hogben G., Dekker M.: Smartphone security: Information security risks, opportunities and recommendations for users, Technical report (December 2010)
13. GSM World, Mobile Privacy (accessed April 15, 2011), http://www.gsmworld.com/our-work/public-policy/mobile_privacy.htm
14. Security on MSNBC, Malware infects more than 50 android apps (accessed April 15, 2011), http://www.msnbc.msn.com/id/41867328/ns/
15. Goguen, J., Mesajue, J.: Security Policies and Security Models. In: Neumann, P. (ed.) Proc. of the 1982 IEEE Symposium on Security and Privacy (SP), pp. 11–20. IEEE Press, USA (1982)
16. Google, Security and Permissions (accessed April 15, 2011), http://developer.android.com/guide/topics/security/Security.html
17. Google, Android Developers (accessed April 15, 2011), http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html
18. RIM, Security overview (accessed April 15, 2011), http://docs.blackberry.com/en/developers/deliverables/21091/Security_overview_1304155_11.jsp
19. RIM, Code Signing Keys (accessed April 15, 2011), http://us.blackberry.com/developers/javaappdev/codekeys.jsp
20. Nokia, Symbian Platform Security Model (accessed April 15, 2011), http://wiki.forum.nokia.com/index.php/Symbian_Platform_Security_Model
21. Nokia, Symbian Signed Test Criteria V4 Wiki version (accessed April 15, 2011), http://wiki.forum.nokia.com/index.php/Symbian_Signed_Test_Criteria_V4_Wiki_version
22. Nokia, Developer_certificate (accessed April 15, 2011), http://wiki.forum.nokia.com/index.php/Developer_certificate
23. Nokia, OVI Publisher Guide, Technical Report (December 2010)
24. Apple, iOS Dev Center (accessed April 15, 2011), http://developer.apple.com/devcenter/ios/index.action
25. Seriot, N.: iPhone Privacy. Black Hat Technical Security Conference, Technical report (February 2010)
26. Microsoft, Windows ® Phone 7 security model, Technical report (December 2010)
27. Microsoft, App Hub (accessed April 15, 2011), http://create.msdn.com/en-US/home/about/developer_registration_walkthrough_confirmation

28. Microsoft, Windows Phone 7 Application Certification Requirements, Technical report, ver. 1.4 (October 2010)
29. Fluxcard, Fluxcard Fake ID (accessed April 15, 2011), http://www.fluxcard.com/
30. GeoTrust, GeoTrust Repository (accessed April 15, 2011), http://www.geotrust.com/resources/repository/legal/
31. GeoTrust, GeoTrust Technical Support (accessed April 15, 2011), https://knowledge.geotrust.com/support/knowledge-base/index?page=chatConsole
32. European Parliament and of the Council of the European Union. Community Framework for Electronic Signatures, Directive 1999/93/EC (December 1999)
33. Legislation.gov.uk, Computer Misuse Act 1990 (accessed April 15, 2011), http://www.legislation.gov.uk/ukpga/1990/18/contents
34. NIST, Secure Hash Standard (SHS), Technical Report FIPS PUB 180-3 (October 2008)
35. Dang, Q.: Recommendation for Applications Using Approved Hash Algorithms. NIST Special Publication 800-107 (February 2009)