Deliverable D6.3.2.9
Usable security for widget sharing

Sini Ruohomaa, Olli Immonen, Marko Lehtimäki

ICT SHOK Future Internet Programme
(ICT SHOK FI)

Phase 2: 1.6.2009 – 31.12.2010

Tivit, Yritysten tutkimus- ja kehittämisrahoitus, Päätös 516/09, 29.5.2009, Dnro 560/31/09

TKK, Tutkimusrahoituspäätös 40212/09, 29.5.2009, Dnro 925/31/09

www.futureinternet.fi

www.tivit.fi

This work was supported by TEKES as part of the Future Internet programme of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

Executive summary / Internal release

Title: Usable security for widget sharing

Widgets, i.e. mobile applications developed by parties other than the platform provider, are distributed through widget sharing systems. Various centralized and distributed measures can be taken to support the user's assessment of these widgets' trustworthiness and to protect the user from security and privacy threats caused by malware. This deliverable analyzes the state of the art in implemented approaches to widget software review, monitoring and trust models.

Content: This deliverable describes the state of the art in implemented approaches to widget software review, monitoring and trust models, studying the best practices in the field. The approaches are evaluated based on their feasibility, costs and security impact.

Impact: We provide an analysis of benefits and disadvantages of implemented widget and relevant other software sharing systems from four categories: centralized software review and certification, distributed software review, user recommendations, and runtime access control. Our overarching research goal is to find usable, feasible and effective methods to support the widget users' evaluation of the trustworthiness of a widget, and to analyze and limit the risk involved in installing and using it. Extracted best practices have been experimented on in D6.3.2.2 (reputation system prototype), and a subset of them has been evaluated through a user study (D6.3.2.8). Although the threat environment of personal mobile devices is rather specialized, the findings can be applied to other software sharing environments as well.

Contact info: Sini Ruohomaa, sini.ruohomaa@cs.helsinki.fi, University of Helsinki, Department of Computer Science; Olli Immonen, olli.immonen@nokia.com, Nokia; Marko Lehtimäki, mzlehtim@cs.helsinki.fi, University of Helsinki, Department of Computer Science

Link: http://www.futureinternet.fi/publications/D6-3-2-9-usable-security.pdf

Usable security for widget sharing

Sini Ruohomaa, Olli Immonen, Marko Lehtimäki

March 31, 2011

Abstract

While mobile phone platforms have been generally tightly closed for years, they are now quickly opening up to resemble small general-purpose computers. The influx of third-party software, widgets, that is not centrally reviewed or governed by the traditional rigid trust models introduces new security risks. These risks must be addressed by the software sharing platform on one hand, and the security-related user interfaces in the mobile phone itself on the other hand. This article presents the state of the art in implemented approaches to widget software review and trust models to extract best practices in the field.

1 Introduction

Mobile phone software platforms have traditionally formed closed ecosystems, where only software endorsed by the platform owner has been possible to install and run on the phone. This limitation has quite effectively protected the user from malicious third-party software, while more open environments, such as PC operating systems, have suffered from different types of malware, spyware and software infected with third-party viruses.

As smartphone capabilities have improved to a point where the devices begin to resemble small general-purpose computers, the closed ecosystem has been opened gradually. This has allowed third party software developers to provide a broad range of applications for the users. Unfortunately, it has also provided malware writers more direct access to user's device (e.g. [21, 38]).

In order to distinguish between platform-provided software and these third-party applications, we call the latter type of software *widgets*: small applications for the mobile phone, with access to the phone's resources, such as address book, persistent storage, GPS tracking capabilities and a camera. While widget software can operate purely locally, many applications are

based on contacting external services, such as photo sharing, instant messaging or a map service. In other words, from a security review perspective, widgets should be seen as dynamic clients to different cloud services rather than local software with a fixed set of code defining its behaviour. In the extreme case, similarly to the principle of Software as a Service, a widget could change its functionality entirely by fetching code from its cloud service dynamically. This kind of software provides extreme flexibility and is always up-to-date, but it is also impossible to review statically, i.e. prior to installation, for any security issues.

As the widget developers are no longer controlled by a single trusted platform provider, the users must evaluate each widget and developer for its trustworthiness separately. From a user's perspective, we define a widget to be trustworthy if it performs the task it advertises, and does not compromise the user's security and privacy by its operation. A widget developer, then, is defined to be trustworthy if he/she produces trustworthy widgets. The Widget Sharing (WiSh) project has studied methods to support a user's trust evaluation of widgets and widget developers, and to limit the risks of installing and using widgets [15, 19, 20, 18, 33].

In order to support the trust evaluation of the user, we can influence two separate environments: the widget distribution service, which is currently still centralized and controlled by the platform provider, or the operating platform for the widgets, running on the mobile device itself. For the purpose of supporting trust evaluations, we assume that the user acquires all their widgets through the widget distribution service[1], i.e. the *widget sharing system*.

The process of acquiring a new widget can be divided into four steps: First, a user accesses the widget sharing system to find a widget they consider interesting and potentially useful. Second, they make a trust decision on whether they want to download the specific widget. If the decision was positive, they download and install it. In the third phase, the user runs the widget on the device, and in the fourth phase of re-evaluation, she may decide to uninstall the widget or keep using it, and/or leave feedback on it for the widget sharing system and the developers to use.

As widget prices are low (or nonexistent) and their installation has been made quite simple, the investment required from a user to try a new widget is small enough to not warrant a very thorough evaluation of whether a widget is actually worth trying out. This reflects on the trust decision as

[1]While this assumption may not hold for the future or for even for current mobile devices that have been "jailbreaked", we expect that the availability of a high-quality distribution service will attract the majority of users and developers, such as with the existing Android Market, Nokia Ovi Store and Apple AppStore.

well: the user cannot not be expected to extend considerable effort in the trustworthiness evaluation on the average, and any additional investment must be justified to her. In other words, the user must be made aware of the security and privacy risks involved in installing and running a widget to motivate due care in making a trust decision, and in running the widget. Our goal, therefore, is to provide usable security support for widgets, with a focus on the widget sharing systems in particular.

The rest of the report is organized as follows: Section 2 reviews existing, implemented solutions and the best practices that can be utilized in the context of widget sharing. Section 3 concludes.

2 Review of solutions

We summarize existing solutions utilizing four different approaches: centralized software review and certification, distributed software review relying on a trusted subset of users, user recommendations, and runtime access control. The amount of effort extended by the widget sharing system operators differs, with the first approach setting the highest demands, and the last fully outsourcing the task to the user and platform manufacturer. Distributed software review relies on a trusted group of users, which must be established and upkept for the system to work, while showing user recommendations without filtering mainly implies a need to exert some control against spam recommendations on the website.

2.1 Centralized software review and certification

Centralized software review and software certification are done by large, globally trusted actors in the widget sharing ecosystem. They form a monolithic core of the trust model, and its main bottleneck, which sets high demands on the efficiency and scalability of the review process. We provide examples from the sofware certification approach of the Symbian platform, and centralized reviewing done in commercial widget sharing systems.

2.1.1 Symbian code signing

The Symbian platform security relies on digital signing and granting capabilities to applications when they are signed [11]. Symbian code signing represents the traditional approach of software certification: in older devices,

unsigned[2] applications could not be installed by default, although this has been enabled since [10].

Signed applications can be distributed through various channels. The signatures can be verified by the software platform at installation time, so the widget sharing system itself does not need to be trusted.

Multiple types of signing are possible. "Express signed" applications have access to a certain range of capabilities, while excluding so-called restricted capabilities [10]. "Certified signed" applications have access even to the restricted capabilities, such as communication device drivers.

Signing test criteria cover such things as successful installation, starting and stopping, no disruption to key device functionalities. They do not cover the application functionality itself [12].

2.1.2 Reviewing in widget stores

Currently, commercial widget sharing is dominated by three stores, catering for different platforms: Nokia Ovi Store, Apple's App Store, and Google's Android Market. Ovi store and Android Market apply a combination of reviewing and runtime access control based on install-time requests for capabilities, also known as manifests; we discuss the manifest approach further in Section 2.4.2.

Nokia Ovi Store requires that all Symbian applications are signed [29]. This way all applications are installed without user prompts, which would be produced by the Symbian platform for unsigned applications. Ovi publishers need to register and pay a small fee. Ovi Store quality assurance inspects and express signs Symbian native and Java applications for free. For certified signing, a third party testing house must be used. There is currently no signing scheme for Symbian/S60 Web Runtime Widgets and Maemo 5 applications.

Apple App Store applications are subject to testing and approval by Apple for basic reliability testing and other analysis [2, 40].

Android Market allows developers to post applications to the software store without initial checking [39]. In the media, the practice has been criticized for potentially leading to security issues [34]. All Android applications are signed by their developers. In this self-signing approach, the signature only serves as evidence that the application was provided by a certain developer, not that it has been verified by any authority. Developers need to register to the store and pay a small registration fee.

[2]When an authorized signature is not available, the developer signs the software themselves: unsigned applications therefore contain self-signatures.

In the Android security architecture [35], applications run in a security sandbox. Applications must declare permissions that they need for additional capabilities. The declaration of requested permissions is made in the application manifest file. The system allows or disallows permission requests based on certificates or by prompting the user. The permissions are declared and approved at install time and are not changed after that. If a widget distributed through the Android Market is deemed malicious and removed from the widget sharing system, it can be remotely uninstalled from users' devices [4] as well, although the option is not automatically applied [21].

2.1.3 Evaluation of overall approach

Benefits:

- Trusted signatures are easy to verify automatically.
- Widget distribution can be done over untrusted servers.

Downsides:

- Static code review and signing do not capture the full functionality of cloud services.
- A centralized reviewer becomes a single bottleneck with a high throughput; this sets high demands for the efficiency of review.
- Reviews are by necessity lightweight: they will catch broken software, but not malicious code (e.g. [21]).
- Any updates to the widget software demand a new review and signature.

We conclude that automated code review can be a valuable tool for widget developers, particularly for highlighting problem areas in code that are difficult to directly test for (such as the energy-efficiency of the software). As the widget sharing ecosystem has opened up, the closed model of installing only approved software is no longer viable as the only option. Trusted monolithic organizations can continue to take advantage of their market status, however, both as high-profile widget developers and as operators of widget sharing systems for other developers.

2.2 Distributed software review

In the distributed software review approach, widget reviewing is delegated from a central actor to a network of trusted volunteer users, which can be recruited into and removed from the trusted group dynamically. The approach is prevalent in open source software development, from which we draw our two example cases: the Debian GNU/Linux distribution, which is a particularly long-standing implementation of the approach, and the Maemo Community, which has more recently applied the same principles on an open source widget sharing system.

2.2.1 Debian GNU/Linux distribution

The Debian GNU/Linux distribution includes a full standalone desktop or server system, from the underlying operating system to applications such as web browsers, games or server software, integrated into a working whole; even a power user may never have to install a piece of software on their machine from outside the distribution set. While a base system fits on an installation CD or DVD, the full Debian distribution contains over 29 000 software packages [7], from various different development teams. The evaluation and repackaging of software into Debian installation packages alone is an effort of over a thousand Debian developers [7]. The underlying project is non-commercial, with a structure codified into a detailed set of policies ranging from high-level decision-making to quality assurance [8].

Every software package included in the Debian distribution is cryptographically signed by a package maintainer [16]. These maintainers are a group of trusted users with registered public keys, and they can add software packages to the central Debian respository. While these users also act as a first defense against clearly malicious code through knowing what and whose software they are submitting, they are not expected to perform a detailed review of the software they package. They do guarantee that the software conforms to a set of standards to ensure it works together with the rest of the software in the distribution.

At first, a software package is placed into a repository for "unstable" software, where early adopters and other developers can test it. If during a certain period of time, no important bugs are filed against it or packages it depends on, the package is passed to the "testing" repository. The testing repository represents the next stable release, and eventually undergoes a schedule to freeze new entries being moved into the repository, final testing and then a release [14]. The stable release is directed at regular users, and bugfixes are released against it as well based on any bugs reported after the

release. All bugs found by the developers and user community are reported in public bug tracker systems.

Ubuntu, a Debian-based distribution, aims to make software quality information more accessible to the end user by providing star ratings and reviews from other Ubuntu users through the package installer software [37]. This approach falls under user recommendations, and we will discuss further examples of it below.

2.2.2 The Maemo Community

The Maemo Community [22] is an open source community with 22.000 registered members and over 900 projects developing software for and around the Linux-based Maemo platform. The Maemo platform is mostly based on open source code, and is used in e.g. the Nokia N810 Internet Tablet and Nokia N900. It has been developed by Nokia in collaboration with many open source projects, including Debian [26].

The software review process is a variant of that used in Debian [23]. First, a user submits a project into a repository called extras-devel. Unlike incorporation to the Debian unstable repository, this step involves no review and no backing from a trusted community member; anyone can set up the Maemo Garage account required for access to this repository. In order to advance to the "beta" repository, extras-testing, the software must go through a basic automated review process, executing a number of basic sanity checks [25].

In order to advance to the stable repository, extras, the software must pass community-based quality assurance. The advancement of the software is determined by voting: 1) at least three members of the testers group, somewhat comparable to Debian packagers, must vote on the software, 2) it must gain 10 more positive votes than negative, and 3) it must have been in the beta repository for at least 10 days. Each version of the software is voted on separately [24].

Through the voting system, the Maemo process partially overcomes the issue of having a smaller community and less time to build a network of trust like the Debian packagers. Users are promoted to the tester group based on their activity in the community, tracked through a site-wide scoring system; the measured activities include testing, development, documentation and discussion. New users only have access to the stable repository; using the other two for anything but testing is strongly discouraged. The stable repository hosts 200 applications which have passed the process [26]. The stable software can be browsed through the Maemo Application Catalog, which presents ratings on a five-star scale and number of downloads as popularity measures.

2.2.3 Evaluation of overall approach

Benefits:

- A community-based review process requires few central resources; it is crowdsourced.
- The target user base for a new software is increased gradually: from the technically savvy early adopters to regular end users.
- In Debian, the initial filter from dedicated, trusted maintainer users keeps obviously malicious software out, limiting the risk of becoming an early adopter.
- In Maemo, the voting system and reviewer promotion model can be implemented with reasonably small and unstructured communities.
- In Maemo, activity scoring creates an incentive for users to participate actively.
- In Maemo, the quality assurance process is partially visible in software distribution, with issue trackers linked on the download pages.

Downsides:

- Interpreting information in a bug tracker system requires expertise.
- The review process is essentially one-way: retractions from the trusted stable repositories are not done lightly, even if security holes are discovered.
- Building a structured community like Debian's takes time; the project has been around since the early 1990s.

We conclude that the distributed review process seems most suitable for a community consisting of majority of technically savvy users, such as the Debian and Maemo user communities. It is unclear how this model might interact with major conflicting commercial interests; reciprocity and retaliation have been observed in electronic marketplaces, such as eBay [30], and sustaining reasonable expert user neutrality might become an issue in a commercial widget sharing environment.

Communicating software quality information is a particularly valuable innovation in the Maemo Community. Summarizing bugtracker information in the widget sharing system user interface in an accessible way would support users' trust decisions: important factors would include whether there are open (particularly security-relevant) issues or not, and how actively open issues are dealt with.

2.3 User recommendations

User recommendations and ratings are a form of feedback on the usability and popularity of an item. They generally do not have anything to do with its security or privacy features, instead focusing on how the software fulfils the functional expectations of users. The format of this feedback ranges from textual comments to numerical feedback, such as "like/dislike" or a star rating on a five-point scale.

When this information is automatically processed to produce recommendations for potentially interesting products, it is known as collaborative filtering [13]; this is a broad topic, with applications areas ranging from movie recommendations (e.g. Movielens [28]) to web pages (e.g. Brin and Page's PageRank [3] for search engines, or Ultra Gleeper [31] for blog-style linking). Recommendations can be formed implicitly as well; on e.g. the Amazon bookstore [1], user behaviour patterns are directly translated to votes: users considering a product are told what other customers interested in it also bought.

In some form, user recommendations are applied by most widget sharing sites. We discuss two cases here that were designed to operate primarily on user recommendations, and lessons learned from them: WidSets, the precursor to Nokia Ovi Store, and Curse.com, a website for distributing extension modules for online games.

2.3.1 WidSets

Nokia WidSets was a community sharing site for AJAX applications for the mobile phone; it was discontinued in 2009 as a part of the migration to Ovi Store as Nokia's main widget sharing system. The web applications in WidSets had very limited capabilities, mostly providing optimized user interfaces for specific RSS feeds or other mobile web browsing, such as accessing Wikipedia articles. WidSets could be accessed either through a web browser or by using a specialized mobile interface.

To support users' searching for new, interesting widgets, the site extended the basic descriptive information on each widget with a range of popularity measures: a counter of how many users liked or disliked the widget, the number of users (equivalent to number of downloads) as well as a sample of the login names of some users using the widget. Different widgets could also be in beta or stable status, indicated with a small diamond logo, and widgets developed by WidSets certified developers had a specially endorsed status.

In the WiSh project, Karvonen et al. conducted a usability study on the WidSets site, and found that the recommendation information was either

poorly understood or underutilized [19]. Users would base their download decision on the widget description, either its rating or number of users (depending on whether they were using the website or the mobile user interface, respectively), and the widget's logo. The results suggest that visual appearance greatly influences users' decisions; a similar phenomenom has been found in the context of web site credibility [9]. Recommendation as well as trustworthiness information should be prominently displayed in order to catch the users' attention. Further, users' awareness of the underlying security issues related to possible malicious widgets and developers was very limited; on the other hand, the widgets in question posed very little threat as well [19].

In further usability studies conducted in the project, covering online recommendation systems including the Nokia Ovi Store [20, 18], signs were found that users evaluating the trustworthiness of a widget there were confused by the overarching Nokia brand of the store. Users expected widgets to be trustworthy, citing their trust in Nokia; they did not seem to consider the widget developer an independent actor. Users may be interpreting Ovi Store like a traditional shop, which carries considerable responsibility over the products it has chosen to sell; this interpretation is problematic from the point of view of activating the users in making more careful trust decisions on individual widgets in the store. More visually prominent aggregation of widget and developer reputation information could alleviate the issue [20, 18].

2.3.2 Curse .com

Curse.com [6] is a community-based site specialized in MMO games (Massively multiplayer online game). It is hosting MMO games focused news and forum services and has significant role as addon distributor for four MMO games. The most popular game is WoW (World of Warcraft) with millions of subscribers. Curse.com is one of the largest distributors for WoW addons.

Addons are small-scale applications that modify game interface and introduce new functionalities to it. In this sense, addons closely resemble smartphone widgets, but have a more limited operating environment.

Curse.com is acting only as an addon distributor. Product development, maintenance, and criticism are done by community members. The site uses a ranking system, where users rate addons on a scale of one to five stars. Users can also freely comment each addon. The site keeps track of addon update dates and supported game versions, and offers detailed information about download rates.

The site uses a category-based sorting system, where each addon may belong to several categories. In addition, the developer can define free keywords

for addons. Based on categories and keywords, the site suggests the most popular similar addons. While browsing an individual addon, the site also recommends other addons generally favored by users with similar interests.

2.3.3 Evaluation of overall approach

Benefits:

- Ratings and reviews are a straightforward form of community involvement, a way to contribute without requiring extensive sustained effort.
- User recommendations support the search for new widgets.

Downsides:

- User recommendation information typically measures popularity, not trustworthiness as such.
- Developers do not form a reputation, only the software does; this becomes problematic particularly when the software can easily be changed.
- Users need a source of motivation for even lightweight contributing.

Recommendations have a particularly strong influence on the widget searching phase, when users are trying to find interesting new widgets to use. They have less weight in trustworthiness decisions. The data format and presentation is particularly important: there are conflicting requirements between keeping the user interface simple, and providing any information that might support the user's decision-making.

Instead of attaching users' experience information only to the end products, i.e. widgets, we expect it would be fruitful to aggregate it to the widget developers. In order to present the aggregated information in a usable way, the developers could be placed into different descriptive categories, such as "newbie developer" (few widgets, recently created account), "established developer" (a high number of downloads, many widgets) or "quality developer" (an established developer with consistently high ratings). For more advanced users, the background data should preferrably be made available when needed.

2.4 Runtime access control

Runtime access control is implemented in the mobile application platform, rather than the widget sharing system. Runtime access management, in its existing forms, disconnects the enforcement process from widget distribution; in theory, the widget sharing system could act as an access policy sharing system, but this is not utilized yet. The manifest approach, in turn, bridges the distribution and runtime platforms in a way that makes it possible to present users with knowledge of the risks of installing the widget already at the widget sharing system, and then enforce their decisions during runtime.

2.4.1 Runtime prompts

Runtime prompts are used to query the user on granting access to a specific resource for a given widget in a fine-grained way. A major downside of the repeated prompts is that they interrupt the user's workflow, and can therefore easily become a nuisance rather than a security tool: to minimize the disruption, users form a reflex to "click ok" on any prompts that pop up, without reading them. This, in turn, nullifies the security effect of the prompt.

The amount of prompting can be reduced by producing compound access policies, such as allowing a given software access to a given resource indefinitely or for a time period. More complex behaviour rules, such as allowing access for a given type of use pattern, or interdependent access rules like "Internet access is not allowed after accessing the address book", may prove more effective, but are also more challenging to configure and and more resource-consuming to monitor. For example Cao and Iverson discuss a set of principles on how to make policy-based access control more usable for the end-user, and propose to implement them through a policy-configuration wizard interface [5].

2.4.2 The manifest approach

In the manifest approach, widget installation packages must include a declaration of their access needs to sensitive resources, such as Internet access or capability to make phone calls. At runtime, it will be granted access only to the resources it has declared to need beforehand, at installation time. The manifest approach has been adopted in some form at least by the Nokia Symbian platform [10, 11] and the Android security architecture [35].

The manifest approach is currently used either to prompt the user at installation time for what kind of access they wish to allow the widget [35],

or also to allow a trusted third party to vouch for the widget's use of the particular capabilities based on a software review [11].

A particular strength of the manifest is that it moves the trust decision about allowing access to resources from the operation of the widget to its installation, where it is less disruptive and might therefore also receive more attention. Using more complex specifications of software behaviour that can be enforced at runtime has been proposed in research as well [27].

While the possibility is not currently utilized, the manifest information could be shown to the user already when they are choosing a widget to download. Combined to an automated risk analysis on the different access types, the user could be presented with concrete examples of what kinds of threats granting a widget this kind of access could entail. A usable presentation of this information is challenging; users should not be overwhelmed with information they are not capable of processing, while more experienced users could benefit from it.

The education of users about risks based on similar data has been proposed by Jacobsson; instead of manifests, Jacobsson automatically analyses End-User License Agreement (EULA) texts to acquire information about potential spyware-like behaviour [17, ch. 5]. While manifests are enforced by the operating platform, EULAs are primarily enforced by courts: developers are motivated to point out e.g. information leaked by the software in order to avoid lawsuit, but they are not required to present the information in a particularly usable way. Visualization of security effects of decisions has been discussed by Tri and Dang [36].

2.4.3 Evaluation of overall approach

Benefits:

- For runtime policies, changes in the software or missed bugs are not an issue.
- Installation-time access to the access requirements is easier to process than runtime requests; could support risk evaluation of installing a widget as well.
- Visible manifests could also encourage developers to think what they really need access to.

Downsides:

- Prompts interrupt the user's work and tend to be ignored as a result.

- Enforcement requires operating system support at runtime, and comes with a computational cost depending on the complexity of policies followed.
- Giving too much weight to manifest-based risk analysis may make it more attractive to produce very narrow-scoped widgets as opposed to general-purpose tools.

Utilizing the manifest information to support the user's risk analysis of downloading a given widget has considerable potential, although making the information accessible is challenging. One possibility would be to actively recommend similar widgets, which have fewer access needs, but this may have undesirable effects on what kinds of widgets developers are then encouraged to make.

One possible approach to easing the usability issue in both policy-setting needs and manifest visualization could be a grouping of commonly needed action/access combinations to profiles. For example "photo/video sharing" needs camera, microphone, possibly GPS and Internet access to a number of specific services, but has no need for dialing out or sending SMSes. High-level profiles can be provided in the user's terminology, avoiding the less accessible technical implementation terms.

3 Conclusion

We have presented the state of the art in supporting users' trust decisions on installing and using widgets. The approaches can be divided into four categories: centralized software review and certification, distributed software review, user recommendations and runtime access control.

The usability of these support functions is important, as all additional effort required on the user's part must be motivated to the user. The user may not be able to see a connection between the access requirements of a widget and a privacy or security threat to herself. In addition to the minimal investment made in decision-making, users are prone to misinterpret the information given, which further emphasises the need to apply usability expertise in designing the user interfaces involved.

The WiSh project has implemented some of the best practices discovered in this review in a prototype reputation system, keeping track of both user experience information and expert-moderated bug reports on widgets, and aggregating this information to produce a form of developer reputation [15, 32]. Initial user experiments on the prototype are reported in another deliverable [33].

Acknowledgements

This research has been done in the Widget Sharing (WiSh) project, which is a collaboration between Nokia, Helsinki Institute of Information Technology (HIIT) and University of Helsinki. The work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

References

[1] The Amazon store and recommendation system website (2007), http://www.amazon.com/

[2] Apple Inc.: App Store Review Guidelines (Mar 2011), http://developer.apple.com/appstore/guidelines.html

[3] Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems 30, 107–117 (Apr 1998), http://dx.doi.org/10.1016/S0169-7552(98)00110-X

[4] Cannings, R.: Exercising our remote application removal feature. Android Developers Blog (Jun 2010), http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html

[5] Cao, X., Iverson, L.: Intentional access management: making access control usable for end-users. In: Proceedings of the second symposium on Usable privacy and security (SOUPS'06). ACM (2006), http://dx.doi.org/10.1145/1143120.1143124

[6] Curse.com website (Mar 2011), http://www.curse.com/

[7] Website of the debian operating system (Mar 2011), http://www.debian.org/

[8] Fernández-Sanguino, J., Garbee, B., Koptein, H., Lohner, N., Lowe, W., Mitchell, B., Murdock, I., Schulze, M., Small, C.: A brief history of Debian, 2.13. Tech. rep., Debian Documentation Team (Sep 2010), http://www.debian.org/doc/manuals/project-history/index.en.html

[9] Fogg, B., Soohoo, C., Danielson, D., Marable, L., Stanford, J., Tauber, E.R.: How do people evaluate a web site's credibility? Tech. rep., Stanford Persuasive Technology Lab (Oct 2002), http://www.consumerwebwatch.org/news/report3_credibilityresearch/stanfordPTL_abstract.htm

[10] Forum Nokia: Capabilities (2011), http://wiki.forum.nokia.com/index.php/Capabilities, accessed 28 March 2011.

[11] Forum Nokia: Packaging and signing (2011), http://www.forum.nokia.com/Distribute/Packaging_and_signing.xhtml, accessed 28 March 2011.

[12] Forum Nokia: Symbian Signed Test Criteria V4 Wiki version (2011), http://wiki.forum.nokia.com/index.php/Symbian_Signed_Test_Criteria_V4_Wiki_version, accessed 28 March 2011.

[13] Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information Tapestry. Communications of the ACM 35, 61–70 (Dec 1992), http://doi.acm.org/10.1145/138859.138867

[14] González-Barahona, J.M., Ortuño Pérez, M.A., de las Heras Quirós, P., Centeno González, J., Matellán Olivera, V.: Counting potatoes: The size of Debian 2.2, revision 0.3a (Jan 2002), http://pascal.case.unibz.it/retrieve/3246/counting-potatoes.html

[15] Hassinen, M., Immonen, O., Karvonen, K., Nurmi, P., Ruohomaa, S.: Trustworthy widget sharing. Tech. Rep. 2010-3, Helsinki Insititute of Information Technology (Dec 2010), https://www.hiit.fi/files/admin/publications/Technical_Reports/hiit-tr-2010-3.pdf

[16] Jackson, I., Schwarz, C., et al.: Debian Policy Manual, version 3.9.1.0 (Jul 2010), http://www.debian.org/doc/debian-policy/

[17] Jacobsson, A.: Privacy and Security in Internet-Based Information Systems. Ph.D. thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden (2008)

[18] Karvonen, K., Immonen, O.: Understanding online reputation information (2011), (unpublished manuscript under submission)

[19] Karvonen, K., Kilinkaridis, T., Immonen, O.: Widsets: A usability study of widget sharing. In: Human-Computer Interaction - Interact 2009. pp. 461–464. No. 5727 in LNCS (2009), http://dx.doi.org/10.1007/978-3-642-03658-3_50

[20] Karvonen, K., Shibasaki, S., Nunes, S., Kaur, P., Immonen, O.: Visual nudges for enhancing the use and produce of reputation information. In: Proceedings of the ACM RecSys 2010 Workshops on User-Centric Evaluation of Recommendation systems and Their Interfaces (UCERSTI).

CEUR-WS.org, Barcelona, Spain (Sep 2010), http://ucersti.ieis.tue.nl/paper1.pdf

[21] Keizer, G.: Google yanks over 50 infected apps from Android Market. NetworkWorld (Mar 2011), http://www.networkworld.com/news/2011/030211-google-yanks-over-50-infected.html, accessed 28 March 2011.

[22] Maemo Community website (Mar 2011), http://www.maemo.org/

[23] Maemo wiki: Extras repository process definition (Aug 2009), http://wiki.maemo.org/Extras_repository_process_definition, accessed 28 March 2011.

[24] Maemo wiki: Extras-testing (Dec 2010), http://wiki.maemo.org/Extras-testing, accessed 28 March 2011.

[25] Maemo wiki: Extras-devel (Feb 2011), http://wiki.maemo.org/Extras-devel, accessed 28 March 2011.

[26] Maemo.org: The Home of the Maemo Community (Mar 2011), http://maemo.org/intro

[27] Massacci, F., Piessens, F., Siahaan, I.: Security-by-contract for the future internet. In: Future Internet - FIS 2008. LNCS, vol. 5468, pp. 29–43 (2009), http://dx.doi.org/10.1007/978-3-642-00985-3_3

[28] MovieLens, a movie recommender system website (2011), http://movielens.umn.edu/

[29] Nokia: Ovi Publisher Guide (Dec 2010), https://p.d.ovi.com/p/g/ovistore_static/docs/Publisher_Guide.pdf

[30] Resnick, P., Zeckhauser, R.: Trust among strangers in internet transactions: Empirical analysis of eBay's reputation system. In: The Economics of the Internet and E-Commerce. Advances in Applied Microeconomics, vol. 11, pp. 127–157. Elsevier Science, Amsterdam (2002), http://www.si.umich.edu/~presnick/papers/ebayNBER/RZNBERBodegaBay.pdf

[31] Richardson, L.: The Ultra Gleeper: A recommendation engine for web pages (Feb 2005), http://www.crummy.com/software/UltraGleeper/IntroPaper.html

[32] Ruohomaa, S., Lehtimäki, M., et al.: Trustworthy widget sharing (Feb 2010), http://www.futureinternet.fi/publications/seminar_2011/wish_seminaariposteri.pdf, poster presented in the Future Internet SHOK results seminar.

[33] Shen, Y., Nurmi, P., Ruohomaa, S., Lehtimäki, M.: D6.3.2.8: Understanding widget downloading preferences. Tech. rep., Tivit Future Internet (Mar 2011)

[34] TechCentral.ie: Spy tool highlights Android app store security issues. TechCentral.ie (Aug 2010), http://www.techcentral.ie/article.aspx?id=15457

[35] The Android Developer's Guide: Security and Permissions (Mar 2011), http://developer.android.com/guide/topics/security/security.html, accessed 28 March 2011.

[36] Tri, D.T., Dang, T.K.: Security visualization for peer-to-peer resource sharing applications. IJCSE 1(2), 47–55 (2009), http://arxiv.org/abs/0912.2289

[37] Ubuntu Wiki: Software Center / Ratings And Reviews (Mar 2011), https://wiki.ubuntu.com/SoftwareCenter/RatingsAndReviews, accessed 28 March 2011.

[38] Vennon, T., Stroop, D.: Threat analysis of the Android Market. Tech. rep., SMobile Systems (Jun 2010), http://globalthreatcenter.com/wp-content/plugins/download-monitor/download.php?id=8

[39] Wikipedia: Android market. Wikipedia (English) (Mar 2011), http://en.wikipedia.org/wiki/Android_Market, accessed 28 March 2011.

[40] Wikipedia: App store. Wikipedia (English) (Mar 2011), http://en.wikipedia.org/wiki/App_store, accessed 28 March 2011.