Trustworthy Widget Sharing

Marja Hassinen, Olli Immonen, Kristiina Karvonen,
Petteri Nurmi, Sini Ruohomaa

HIIT Technical Reports 2010-3

Contact Information

Helsinki Institute for Information Technology HIIT
Tietotekniikan tutkimuslaitos HIIT (in Finnish)
Forskningsinstitutet för Informationsteknologi HIIT (in Swedish)

hiit-info@hiit.fi
www.hiit.fi

HIIT Otaniemi Site

Postal address:
Helsinki Institute for Information Technology HIIT
PO Box 19215
00076 Aalto, Finland

Street address:
Technopolis Innopoli 2
Tekniikantie 14, Espoo

Telephone: +358 9 47001
Fax: +358 9 694 9768

HIIT Kumpula Site

Postal address:
Helsinki Institute for Information Technology HIIT
PO Box 68
00014 University of Helsinki, Finland

Street address:
University of Helsinki, Department of
Computer Science, Exactum
Gustaf Hällströmin katu 2b, Helsinki

Telephone: +358 9 1911
Fax: +358 9 191 51120

HIIT Technical Reports 2010-3
ISBN 978-952-60-3543-7 (electronic)
ISSN 1458-9478 (electronic)

Contents

Chapter 1

Introduction and structure

This report contains the deliverables of the first Widget Sharing (WiSh) project. The project begun as a collaboration between Nokia, HIIT and University of Helsinki in 2007. In 2008–2010 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).

As Nokia introduced widget support for the S60 mobile phone platform, it became possible to create widgets using familiar standards-based Web technologies available to all S60 licensees. To facilitate the creation of a wide selection of innovative widgets, an open and non-bureaucratic development market was found beneficial. The Widgets Sharing (WiSh) project was launched to do research on finding and evaluating a suitable trust model for realising a reputation management system for sharing widgets provided by an open community. The qualitative objective of the reputation system is to encourage users to install and use new widgets by providing a sense of trust.

Widgets are web applications which use the browser as an environment to run in, but which do not have the usual browser chrome. They are installed on a local client, and have several features that are not available to regular web applications, such as a more explicit security model to allow them to perform a wider range of tasks easily, and a persistent information store apart from cookies. To make the benefit of a trust model for Widgets clearer, there were two concrete requirements:

1. to allow Widgets access to OS and application resources, such as contacts, GPS and the ability to send and receive messages.

2. To allow Widgets to access resources on the Web, one or several identified hosts, or even any host.

Both these requirements make it necessary to to loosen up the security model of Web browsers, sandboxing, which disallows a Web application access

to most local resources and restrict downloading and accessing JavaScript code from other hosts than the Web application's host of origin.

The project team members of the first phase reported here were Kristiina Karvonen (HIIT, project leader), Sini Ruohomaa (UH), Petteri Nurmi (HIIT) and Marja Hassinen (HIIT). Olli Immonen represented Nokia in the project. The steering group included Kristiina Karvonen, Martti Mäntylä, Lea Kutvonen, Olli Immonen, Guido Grassel, Patrik Floréen and Antti Ylä-Jääski. In later phases, Theofanis Kilinkardis, Yiyun Shen, Olli Niinivaara, Taru Itäpelto, Tania Nunes, Sanna Shibasaki, Puneet Kaur, Marko Lehtimäki, Aleksi Hankalahti and Aurora Tulilaulu have also participated in the project as research assistants. Aleksi Hankalahti has been responsible for editing this compilation. The authors would like to gratefully acknowledge their contributions to the project.

Further publicly available reports from the later phases of the project include, at the time of writing, a conference publication [KKI09] and a Master's thesis [She10].

The report is structured as follows: Chapter 2 presents the preliminary concept model work, including use and misuse cases and threat analysis for the designed widget sharing system. The finalized concept model presented in Chapter 3 specifies the reputation and risk management mechanisms in order to support desired incentives of users and developers. Chapter 4 presents evaluation plans of the widget sharing system, which have served as a guideline for work in later phases of the project. Finally, Chapter 5 concludes the report.

Chapter 2

Preliminary concept model

This chapter consists of four parts:

2.1 Use cases

2.2 Experimental use cases

2.3 Threat analysis

2.4 Misuse cases

Use cases lists and describes use cases for different roles in the system: a widget user (logged in or anonymous), a widget developer, a moderator, who enforces rules of proper conduct in the system, and an administrator, who can perform rare administrative activities. The dividing line between the duties of the moderator and administrator roles is frequency: day-to-day management, such as the handling of abuse reports, is delegated to moderators, while administrators watch over moderators and can perform less frequently needed tasks which require extensive access rights.

Experimental use cases describes interesting possible use cases that add new layers of complexity to the system, but may be worth exploring further. The two main threads here are a "friends" system for taking advantage of users' social networks in widget recommendations, and expert reviewing. Social networks can be used both between regular users to spread information about interesting widgets, and between developers: newcomers can gain reputation for themselves and their widgets through explicit support from more established developers. Expert reviewing aims to provide more and higher-quality information than a general user rating, involving a deeper analysis of a widget, its capabilities and trustworthiness. This analysis should be done by users who can be considered sufficiently qualified to evaluate widgets, but also neutral enough to provide a truthful evaluation. This topic

also covers giving special acknowledgement to good developers or widgets, either due to fulfilling a given set of standards, or to express a more general sense of goodness.

Threat analysis is a general analysis of different actors' assets in the system, and threats towards them caused by other actors, including the system itself. This also includes threats not caused by actual malice, but by e.g. contradicting goals between users. The analysis considers different actors through their roles, in order to differentiate between threats caused by different types of activities, such as developing a widget, using it and rating it. Certain threats are activated by misbehaviour, attacks or other unintended use of the system.

Misuse cases presents attacks and other unintended uses of the system that activate threats. The section is divided into two parts: The first part focuses on widget capabilities, and lists different types of misuse that is enabled by granting the widget certain types of access rights; the risks are present whether the widget has been designed to misbehave by a malicious developer, or if unintentional bugs or vulnerabilities either cause it to misbehave or allow it to be controlled by a malicious third party. The second part lists and describes various kinds of unwanted behaviour by users in the widget sharing system itself; the main targets of misuse involve compromising the reputation and abuse reporting systems.

2.1 Use cases

This section describes the use cases of the Widget Sharing system.

Roles in the system:

User downloads widgets and gives feedback

Developer uploads widgets

Moderator reads abuse reports, can lock users / widgets

Administrator can cancel moderators' actions and grant moderator privileges

One user can have multiple roles. That is, an administrator is also a moderator, a moderator and a developer are also users. (A moderator may be or may not be a developer.) Moderators can be volunteer users; recruitment methods have been left for the administrator to decide.

The user role is available to all registered users. The developer role becomes available to the user when he uploads his first widget (use case Developer-001:

Uploading a new widget (Section 2.1.3, page 20)). The moderator role becomes available to the user when an administrator grants him moderator access (use case Administrator-001: Granting moderator status to a user (Section 2.1.5, page 31)). Access to the administrator role is granted out-of-band. Use cases typically require access to the role, with the exception of Developer-001: Uploading a new widget (Section 2.1.3, page 20) (prerequisite for assuming the developer role for the first time). Users can also be anonymous or logged in.

The structure for describing use cases is as follows:

Actor-Number: Use case name

Preconditions
: Conditions necessary to carry on this use case (e.g., examining a certain view of the system: the user must view a widget's page before downloading the widget).

Use cases before
: Use cases to be performed before this use case.

Use cases after
: Possible use cases which can be performed after this use case.

Normal progress
: The steps taken by the actor to go trough the use case, and information shown to the user.

Exceptional conditions
: Possible error conditions etc. related to the use case.

2.1.1 List of use cases for different roles

Use cases for the user:

- User-001: Registering into the system (Section 2.1.2, page 7)
- User-002: Unregistering from the system (Section 2.1.2, page 8)
- User-003: Logging in (Section 2.1.2, page 9)
- User-004: Logging out (Section 2.1.2, page 10)
- User-005: Updating own profile (Section 2.1.2, page 10)
- User-006: Browsing widgets (Section 2.1.2, page 11)
- User-007: Searching widgets (Section 2.1.2, page 12)

- User-008: Examining a widget (Section 2.1.2, page 12)
- User-009: Examining a user's profile (Section 2.1.2, page 14)
- User-010: Downloading a widget (Section 2.1.2, page 16)
- User-011: Rating a widget (Section 2.1.2, page 16)
- User-012: Sending an abuse report (Section 2.1.2, page 17)
- User-013: Searching users (Section 2.1.2, page 18)
- User-014: Using a widget (Section 2.1.2, page 19)

Anonymous users (not registered or logged in) have access to the following use cases:

- User-001: Registering into the system (Section 2.1.2, page 7)
- User-003: Logging in (Section 2.1.2, page 9)
- User-006: Browsing widgets (Section 2.1.2, page 11)
- User-007: Searching widgets (Section 2.1.2, page 12)
- User-008: Examining a widget (Section 2.1.2, page 12)
- User-009: Examining a user's profile (Section 2.1.2, page 14)
- User-010: Downloading a widget (Section 2.1.2, page 16)
- User-013: Searching users (Section 2.1.2, page 18)
- User-014: Using a widget (Section 2.1.2, page 19)

Note: as default we assume that User-011: Rating a widget (Section 2.1.2, page 16) is not available to unregistered users.

Use cases for the widget developer:

- Developer-001: Uploading a new widget (Section 2.1.3, page 20)
- Developer-002: Updating an existing widget (Section 2.1.3, page 21)
- Developer-003: Removing a widget (Section 2.1.3, page 22)
- Developer-004: Restoring a removed widget (Section 2.1.3, page 23)
- Developer-005: Verifying co-authorship (Section 2.1.3, page 24)

Use cases for the moderator:

- Moderator-001: Reading an abuse report (Section 2.1.4, page 25)
- Moderator-002: Locking a widget (Section 2.1.4, page 27)
- Moderator-003: Locking a user (Section 2.1.4, page 27)
- Moderator-004: Unlocking a widget (Section 2.1.4, page 28)
- Moderator-005: Unlocking a user (Section 2.1.4, page 29)
- Moderator-006: Removing feedback (Section 2.1.4, page 30)
- Moderator-007: Restoring feedback (Section 2.1.4, page 30)

Use cases for the administrator:

- Administrator-001: Granting moderator status to a user (Section 2.1.5, page 31)
- Administrator-002: Removing moderator status from a user (Section 2.1.5, page 31)
- Administrator-003: Removing a widget permanently from the system (Section 2.1.5, page 32)
- Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33)
- Administrator-005: Restoring a user to the system (Section 2.1.5, page 34)

The reputation / trust mechanism (not considered here) may introduce additional use cases. See Section 2.2.

2.1.2 Use cases for the user

User-001: Registering into the system

Preconditions
None. This use case is a prerequisite for logging into the system and through that a prerequisite for all actions requiring login.

Use cases before
N/A

Use cases after

User-003: Logging in (Section 2.1.2, page 9), User-005: Updating own profile (Section 2.1.2, page 10).

Normal progress

1. The user fills the following information into a registration form:
   - user name (unique),
   - password, and
   - e-mail address.
2. The information is verified and an account is created for the user.
3. The user is automatically logged in for the first time and directed to update his profile, see User-005: Updating own profile (Section 2.1.2, page 10).

Exceptional conditions

If a user by the same name exists already (or has existed recently) or other compulsory information is not provided, a new account is not created and the user notified of the error.

Notes

- Other compulsory information may include a valid email address or another means of connecting the user to a real world person. Requiring more information from all users increases the threshold for registration.
- Using "CAPTCHAs" (symbols the user has to type)?
- A verification phase: an e-mail is sent to the user and the user has to click a link to make the account work?

User-002: Unregistering from the system

Preconditions

The user must have registered into the system, see User-001, and be logged in, see User-003.

Use cases before

N/A

Use cases after

N/A

Normal progress

1. The user indicates and confirms that he wants to unregister from the system, i.e. remove his user account permanently.
2. The user is automatically logged out (see User-004: Logging out (Section 2.1.2, page 10)).
3. The user's account is removed. Any widgets the user has uploaded alone are removed, and widgets co-developed by the user no longer list the user as one of the developers (or indicate that the user account is removed). If a widget has no remaining unremoved developers as a result, it is removed as well. The user's moderation activities are not cancelled; restoring can still be done by an administrator.

Exceptional conditions

N/A

Notes

- For developers, there may be a need to e.g. make user removal cancellable for a given period of time in case of account hijacking or moments of bad decision-making.
- Should there be a restriction that a user name cannot be identical to any other user name that has ever existed in the system?
  - What about users who create multiple user accounts to make the available the user name space smaller?

User-003: Logging in

Preconditions

The user must have registered into the system, see User-001. This use case is a prerequisite for all other actions except User-001 (registering) and anonymous browsing/downloading (User-006 through to User-010).

Use cases before

N/A

Use cases after

N/A

Normal progress

1. The user enters his user name and password into a login prompt, OR the user's browser provides alternative login information e.g. through a cookie. ("Remember my login.")

2. The user's state changes from anonymous to logged in, and he gains access to use cases of all other roles he can assume. (See role descriptions above.)
3. The user's status (being online) may be shown in his profile.

Exceptional conditions

If the username or password entered do not match, logging in fails and the user is notified. If the user has forgot his login or password, the user is offered a way to find them out.

User-004: Logging out

Preconditions

The user must be logged in (see User-003: Logging in (Section 2.1.2, page 9)).

Use cases before

N/A

Use cases after

N/A

Normal progress

1. The user logs out by clicking on a "log out" -button or by unregistering from the system (User-002: Unregistering from the system (Section 2.1.2, page 8)) (alternatively by remaining inactive for a given period (e.g. a month) or by removing/changing session information e.g. by closing the browser).
2. The user's state changes from logged in to anonymous, and his access to use cases is limited to anonymous browsing, registering and logging in (See role descriptions above.)
3. The user's status (being offline) may be shown in his profile.

Exceptional conditions

N/A

User-005: Updating own profile

Preconditions

N/A

Use cases before

User-001: Registering into the system (Section 2.1.2, page 7) or User-009: Examining a user's profile (Section 2.1.2, page 14).

Use cases after

N/A

Normal progress

1. The user can modify the following social information:
   - name,
   - picture,
   - description,
   - contact information, and
   - password.
2. The user clicks "Submit".
3. If the update is successful, the system displays the new profile.

Exceptional conditions

The user didn't give a valid password (if required): The system displays the editing page and an error message.

User-006: Browsing widgets

Preconditions

N/A

Use cases before

N/A

Use cases after

User-008: Examining a widget (Section 2.1.2, page 12).

Normal progress

1. The user selects a widget category, e.g., messaging / weather / utilities / fun and possibly a subcategory.
2. The user sees a list of widgets.
3. Each widget has the following information on display:
   - widget name and an icon,
   - description (or beginning of the description),
   - widget developers' id's (which are links to user profiles),

- adding time of the widget,
- number of downloads and average rating, and
- overall trustworthiness assessment.

4. The user has a possibility to click the widget and access the widget's profile.

Exceptional conditions

N/A

Notes

- Information to display on a mobile device?
- Ordering of the widgets? Popularity / trustworthiness / newer first?

User-007: Searching widgets

Preconditions

N/A

Use cases before

N/A

Use cases after

User-008: Examining a widget (Section 2.1.2, page 12).

Normal progress

1. The user enters a search condition:
   - partial name of the widget,
   - a word occurring in the description of the widget (e.g., "news"), and/or
   - partial name of a developer?
2. The user sees a list of widgets, see User-006: Browsing widgets (Section 2.1.2, page 11).

Exceptional conditions

N/A

Notes

- Ordering of the widgets? Match accuracy / popularity / trustworthiness / newer first?

User-008: Examining a widget

Preconditions

The user has found a widget's page through browsing widgets, searching widgets or by link (e.g., from a developer's profile).

Use cases before

User-006: Browsing widgets (Section 2.1.2, page 11), User-007: Searching widgets (Section 2.1.2, page 12), User-009: Examining another user's profile (Section 2.1.2, page 14) (all optional).

Use cases after

User-010: Downloading a widget (Section 2.1.2, page 16), for developers: Developer-002: Updating an existing widget (Section 2.1.3, page 21), Developer-003: Removing a widget (Section 2.1.3, page 22).

Normal progress

1. The user sees the following information about the widget:
   - General information (provided by the widget developers):
     - name of the widget and an icon, and
     - a description of the widget (+ categories the widget belongs to?).
   - Rating information:
     - If the user has given a rating / commented the widget, the rating and the comment are shown. The name of the rater is a link to his profile.
   - Information for assessing trustworthiness quickly:
     - overall trustworthiness assessment and risk visualization, and
     - statistics: number of downloads, (weighted) average rating, abuse reports?
   - Further information to aid the assessment:
     - a manifest: which resources does the widget use (suitably visualized);
     - information about the author: name / identifier, statistics: number of widgets uploaded, their names + icons, number of downloads and average ratings;
     - the user can click links to visit the widget developers' profiles; and
     - all ratings and textual comments.
   - Upkeep information:

- If the user fulfils the prerequisites for User-011: Rating a widget (Section 2.1.2, page 16), he is provided an opportunity to add a new rating/comment, or modify his earlier rating and give a new comment.
- If the user is one of the developers of the widget, he is provided an opportunity to update the widget (see Developer-002: Updating an existing widget (Section 2.1.3, page 21)) or remove it (see Developer-003: Removing a widget (Section 2.1.3, page 22)).

2. The user assesses the trustworthiness and usefulness of the widget and decides whether the widget is worth downloading.

Exceptional conditions

N/A

Notes

- The average rating could be weighted by taking the reputation / amount of contribution of the raters into account.
- Some parts of widget information, e.g., the ratings and textual comments, may be hidden so that the user can access them only by clicking a link.

User-009: Examining a user's profile

Preconditions

The user is viewing a user's profile.

Use cases before

For finding a user profile: User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The user sees the following information:
   - Social information:
     - identifier,
     - name,
     - picture,
     - description, and

    - contact information.
  - Information for assessing credibility / reputation:
    - status: online / offline,
    - last login, registered since,
    - moderator / administrator status,
    - possible "locked" status (if the user has been locked by a moderator or an administrator),
    - statistics: number of ratings given, average rating,
    - rated widgets, their ratings and textual comments, and
    - possibly downloaded widgets.
2. If the viewed user is also a developer, the following information is also shown:
  - A list of widgets developed:
    - widget name and an icon (which are links to the widget's page),
    - description (or beginning of the description),
    - widget developers' id's (which are links to user profiles),
    - adding time of the widget,
    - number of downloads and (weighted) average rating, and
    - overall trustworthiness assessment.
3. If the user is viewing his own profile, he has a possibility to click "Update" — or should the possibility to update be there without clicking anything?
4. If the user is a developer and is viewing his own profile, the following information is shown:
  - A list of removed widgets:
    - widget name and an icon (which are links to the widget's update / restoring page),
    - description (or beginning of the description),
    - widget developers' id's (which are links to user profiles),
    - adding time of the widget, removal time of the widget,
    - number of downloads and average rating, and
    - overall trustworthiness assessment.
5. If the user is a moderator, he has a possibility to proceed to use case Moderator-003: Locking a user (Section 2.1.4, page 27) (if the user is not locked) or Moderator-005: Unlocking a user (Section 2.1.4, page 29) (if the user is locked).

Exceptional conditions
N/A
Notes

- The average rating could be weighted by taking the reputation / amount of contribution of the raters into account.
- Should some information be displayed only non-anonymous users (users who are currently logged in)?
- Ratings and comments may be useful for assessing the credibility of a user. E.g., if the user always complains that the widgets misbehave, other users may decide not to trust these comments.
- Do we want to keep the information about downloaded widgets private so that a user cannot see which widgets another users have downloaded? (Or let the users decide whether downloaded widgets are shown on their profiles?)

User-010: Downloading a widget

Preconditions
The user is viewing a widget's page and the widget is not locked.
Use cases before
User-008: Examining a widget (Section 2.1.2, page 12).
Use cases after
N/A
Normal progress

1. The user clicks "Download".
2. The system starts downloading the widget to the user's system.

Exceptional conditions
N/A

User-011: Rating a widget (or modifying an existing rating)

Preconditions
The user is viewing a widget's page.
Use cases before
User-008: Examining a widget (Section 2.1.2, page 12), possibly: User-010: Downloading a widget (Section 2.1.2, page 16).

Use cases after

N/A

Normal progress

1. The user sees the widget's page (see (Section 2.1.2, page 12)) and decides to leave feedback (or modify the given rating).
2. The user enters a (new) rating and a textual comment (optional). The rating describes how entertaining / useful the widget is. The user is not required to assess the safety / security of the widget.
3. The user clicks "Submit".
4. The system displays the new rating and the comment. (Or the widget's page including these?) If the user has rated the widget before, the old rating is replaced. (The textual comment is not replaced.)

Alternatively, the user can do this while using the widget, through a mobile rating interface provided by the widget platform.

Exceptional conditions

N/A

Notes

- Old and new ratings / comments are visible, but only the newest rating is taken into account when calculating means etc.
- The user is not required to assess (capable of assessing) the safety / security of the widget. The widget might be malicious but also seem to be entertaining / useful.
- How to integrate giving ratings as a natural part of user experience?
  - Integrating the possibility to give feedback into the Widget Platform.
  - Could the platform ask the user to give feedback if the user has not given it?
  - This use case could be performed automatically by the widget platform: with the widget icon the user could see possibility for entering stars, then when the user clicks the rating, the widget platform could automatically log in and give the rating.
- Privacy of ratings and comments: Here we have assumed that ratings and comments are public and if a user rates a widget, he also makes public that he is using it.

User-012: Sending an abuse report

Preconditions
The user is viewing a widget's page and thinks that the widget is dangerous

Use cases before
User-008: Examining a widget (Section 2.1.2, page 12)

Use cases after
N/A

Normal progress

1. The user clicks "Send abuse report".
2. A text field appears and the user enters a description of the abuse (e.g., this widget is phishing credit card numbers).
3. The user clicks "Send".
4. The system shows the user a message saying that the report was sent.
5. The abuse report can then be accessed by moderators and administrators from a abuse report log.

Exceptional conditions
N/A

Notes

- To prevent revenge, abuse reports are private so that it is not visible that a user has sent an abuse report.
- Should something happen automatically when an abuse report is sent?
  - A notification on the widget's page is not very useful if the user's name is not shown: the user may be a misbehaving user who sends a lot of abuse reports.

User-013: Searching users

Preconditions
N/A

Use cases before
N/A

Use cases after

User-008: Examining a user's profile (Section 2.1.2, page 14).

Normal progress

1. The user enters a search condition:
   - partial name / id of the user
   - other possibilities?
2. The user sees a list of users matching the query. The names are links to the users' profiles. The list includes the following information:
   - identifier and name, and
   - measure of "famousness" in the system? Number of widgets developed and their number of downloads and average ratings, names of co-developers?

Exceptional conditions

N/A

Notes

- Information shown on a mobile interface? Only names and identifiers?
- Possible search conditions depend on what information is inserted into a user's profile.

User-014: Using a widget

Preconditions

The user has a Widget Platform and has downloaded at least one widget.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12), User-010: Downloading a widget (Section 2.1.2, page 16).

Use cases after

Possibly: User-011: Rating a widget (Section 2.1.2, page 16), User-012: Sending an abuse report (Section 2.1.2, page 17), User-009: Examining a user's profile (Section 2.1.2, page 14), User-010: Downloading a widget (Section 2.1.2, page 16).

Normal progress

1. The user starts the widget by clicking it.
2. If there are updates available, the Widget Platform displays a message and enables the user to examine and download the newest version.
3. If the widget is locked / removed in the Widget Sharing system, the Widget Platform displays a warning.
4. The user has also possibility to give feedback easily (e.g., entering a rating and a textual comment which are then added to the Widget Sharing system by the Widget Platform). This is an alternative interface for User-011: Rating a widget (Section 2.1.2, page 16).

Exceptional conditions

- If the widget is locked / removed from the Widget Sharing system, the Widget Platform might make the widget unusable?

Notes

- The Widget Platform can encourage the user to rate the widget if the user has not already done so. The question remains, how to do it so that the users will not find it annoying.

2.1.3 Use cases for the developer

Developer-001: Uploading a new widget

Preconditions
The developer is logged in.
Use cases before
N/A
Use cases after
User-008: Examining a widget (Section 2.1.2, page 12).
Normal progress

1. The developer clicks "Upload new widget"
2. The developer enters the following information:
   - name,
   - icon (a file containing the icon, to be uploaded),

- other developers of the widget (their id's in the system),
- textual description of the widget,
- categories for the widget (e.g., messaging / utilities / fun),
- a manifest describing which resources the widget uses, and
- source code of the widget (a file / files containing the source code, to be uploaded).

3. The developer clicks "Send".
4. If the upload was successful, the system displays a message stating a notification is sent to other developers. When they verify that they are co-developers of the widget (see Developer-005: Verifying co-authorship (Section 2.1.3, page 24)), the widget is added. If there are no co-developers, the widget is added immediately and the widget's page is shown.

Exceptional conditions

The developer does not enter all compulsory information / enters invalid information (e.g., invalid developer id's): The system shows the upload page again with error messages.

Notes

The "Upload new widget" -button may be visible all the time on the desktop interface — but where is it located on a mobile interface?

Developer-002: Updating an existing widget

Preconditions

The developer is logged in and is viewing a widget's page. The developer is one of the developers of the widget.

Use cases before

User-006: Browsing widgets (Section 2.1.2, page 11), User-007: Searching widgets (Section 2.1.2, page 12), User-009: Examining another user's profile (Section 2.1.2, page 14).

Use cases after

User-008: Examining a widget (Section 2.1.2, page 12).

Normal progress

1. The developer has the possibility of modifying the following information:
   - name,
   - icon (a file containing the icon, to be uploaded),

- other developers of the widget (their id's in the system),
- textual description of the widget,
- categories for the widget (e.g., messaging / utilities / fun),
- a manifest describing which resources the widget uses, and
- source code of the widget (a file / files containing the source code, to be uploaded).

2. The developer clicks "Send".
3. If the date was successful, the system displays a message stating a notification is sent to other developers. If new developers are added, they must verify their co-authorship (see Developer-005: Verifying co-authorship (Section 2.1.3, page 24)).

Exceptional conditions

The developer enters invalid information (e.g., invalid developer id's): The system shows the update page again with error messages.

Notes

- It's probably not a good idea to require that all developers accept the changes — some may be inactive in the system, e.g., they do not develop widgets any more.
- What if a developer removes other developers and gains a control over the widget? What are the rights of removed developers? Or is it impossible to remove developers other than himself? What if the update includes a new version and only a subset of the developers have contributed and others are inactive?
- This use case is also used for defining that a developer does not want to be a widget author any more (e.g., if he has not contributed to the current version). What if all developers do it? Is the widget them removed?
- The Widget platform might check if there are updates available and inform the user.
  - Other notifications to users who have downloaded the widget?

Developer-003: Removing a widget

Preconditions

The developer is logged in and is viewing a widget's page. The developer is one of the developers of the widget.

Use cases before

User-006: Browsing widgets (Section 2.1.2, page 11), User-007: Searching widgets (Section 2.1.2, page 12), User-009: Examining a user's profile (Section 2.1.2, page 14).

Use cases after

N/A

Normal progress

1. The developer clicks "Remove widget."
2. The widget is removed from the system and the system displays a message stating a notification is sent to other developers.
3. The developers can see the widget by examining their own profiles but otherwise the widget is not visible in the system.

Exceptional conditions

N/A

Notes

- It's probably not a good idea to require that all developers accept the removal — some may be inactive in the system, e.g., they do not develop widgets any more.
- What if a developer removes a widget against other developers' will? (Let them handle the situation in real life?)
- Is some other information about the removed widget still visible in the system?
- The Widget Platform might inform the user if his widgets are removed from the Widget Sharing system.

Developer-004: Restoring a removed widget

Preconditions

The developer is logged in and has examined his profile, found a removed widget there and clicked it. Now the developer is viewing a update page for the removed widget.

Use cases before

User-009: Examining a user's profile (Section 2.1.2, page 14).

Use cases after

N/A

Normal progress

1. The developer can update the widget's information, see Developer-002: Updating an existing widget (Section 2.1.3, page 21).
2. The developer clicks "Restore widget."
3. The widget is restored to the system and the system displays a message stating a notification is sent to other developers.

Exceptional conditions

N/A

Notes

- It's probably not a good idea to require that all developers accept the restoring — some may be inactive in the system, e.g., they do not develop widgets any more.
- What if a developer restores a widget against other developers' will? (Let them handle the situation in real life?)

Developer-005: Verifying co-authorship

Preconditions

The developer is logged in is examining his notifications or his profile.

Use cases before

User-009: Examining a user's profile (Section 2.1.2, page 14) (optional).

Use cases after

N/A

Normal progress

1. The developer sees that there is a notification of co-authorship.
2. The developer clicks the notification to see further information of the widget in question:
   - name,
   - icon,
   - uploader of the widget,
   - other developers of the widget (their id's in the system) and if they have verified the co-authorship,
   - description of the widget, and

- categories for the widget (e.g., messaging / utilities / fun).

3. If the developer thinks that he is a co-author, he clicks "I'm a co-author". The system shows a message saying that the notification is accepted. If all developers have now verified their co-authorship, the widget is added to the system.
4. If the developer thinks he is not a co-author, he clicks "I'm not a co-author". The developer is removed from the author list of the widget.

Exceptional conditions

N/A

Notes

- Are people who "try phishing co-authorships" of famous developers punished somehow?

2.1.4 Use cases for the moderator

Moderator-001: Examining an abuse report

Preconditions

A user has sent an abuse report (User-012: Sending an abuse report (Section 2.1.2, page 17)).

Use cases before

N/A

Use cases after

Possibly: User-008: Examining a widget (Section 2.1.2, page 12) or User-009: Examining a user's profile (Section 2.1.2, page 14), Moderator-002: Locking a widget (Section 2.1.4, page 27) or user Moderator-005: Locking a user (Section 2.1.4, page 27).

Normal progress

1. The moderator enters a page listing abuse reports. For each abuse report in the list, the following information is shown:
   - Report header: An identifier for the abuse report (see User-012: Sending an abuse report (Section 2.1.2, page 17)), time of reporting, reporter, target widget.
   - The status of the report: new, processing, resolved.

- If the status of the report is "processing" or "resolved", the name of the moderator who last modified the state of the report, and the last modification date is shown.
- The reports can be sorted by time of reporting or state modification, reporter, target widget, status and moderator. (Some of this information may be hidden for the mobile user interface and accessible through alternative means, such as hovering/clicking.)

2. The moderator chooses an abuse report to examine more carefully. The following more detailed information is shown about the abuse report:
   - The report header (see above).
   - The status of the report and the date of the last modification to the report, and the moderator who did the modification.
   - The user comment on the abuse report.
   - Comments by moderators, if they have been made.
   - All mentions of users and widgets act as links to user profiles and widget pages (use cases User-009: Examining a user's profile (Section 2.1.2, page 14) and User-008: Examining a widget (Section 2.1.2, page 12) respectively). The moderator may use them to move to use case Moderator-002: Locking a widget (Section 2.1.4, page 27) and/or Moderator-005: Locking a user (Section 2.1.4, page 27) at this point.
3. The moderator updates the abuse report (or stops the use case):
   a) The moderator changes the status of the report and adds a comment to it. Particularly, if the moderator closes the report (changes the status to resolved), he must provide an explanation of what was done: the report was determined to be spam (i.e. frivolous reporting) or a false alarm (i.e. the widget seems ok, but the report expressed a valid concern), the widget has been locked, and possibly in addition either the developer(s) of the widget or the frivolous reporter user accounts have been locked. By changing the status to "processing" the moderator can indicate he is working on the report, but needs some time to research. All moderators can modify the status of an abuse report, also to reopen the report by changing the status from resolved to new or processing; or
   b) The moderator adds a comment without changing the status of the report.

Exceptional conditions

N/A

Notes

While this moderator action is possible to do over a mobile user interface, it probably not typically done "on the road" since it requires concentration.

Moderator-002: Locking a widget

Preconditions

The moderator is examining a widget and thinks that the widget is dangerous or inappropriate, possibly after reading an abuse report on the widget.

Use cases before

Moderator-001: Reading an abuse report (Section 2.1.4, page 25) (not necessary), User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The moderator clicks a button "Lock this widget" and describes the reason for locking the widget.
2. The widget is marked as locked and reason for locking and the locker id are included in it's page. It cannot be downloaded or rated. It cannot be found in the widget category listing but its page can be accessed via a direct link or searching it by name.
3. The developers of the locked widget are notified (e.g., by sending an e-mail). Also the administrators and other moderators are notified (e.g., showing the locking in a log of recent moderator activity).

Exceptional conditions

N/A

Notes

- The Widget running platform might check if some of the user's widgets are locked and inform the user.
- This moderator action is unlikely to be done through the mobile user interface, since analysing a widget requires concentration.

Moderator-003: Locking a user

Preconditions

The moderator is examining a user and thinks that the user is malicious or misbehaving.

Use cases before

Moderator-001: Reading an abuse report (Section 2.1.4, page 25) and User-008: Examining a widget (Section 2.1.2, page 12) (not necessarily), User-009: Examining a user's profile (Section 2.1.2, page 14).

Use cases after

N/A

Normal progress

1. The moderator clicks a button "Lock this user" and describes the reason for locking the user. The moderator can also enter a date for automatic unlocking (e.g., a punishment for spamming the abuse reports might be a 1 month ban from the Widget Sharing system).
2. The user is marked as locked and a notification describing a reason for locking is sent to him. Also the administrators, and other moderators are notified (e.g., showing the locking in a log of recent moderator activity).
3. The user's profile is frozen so that only the identifier and status (locked) are shown.
4. The user can log in and if the locked user is a developer, he can remove his own widgets. Other use cases are not available.

Exceptional conditions

N/A

Notes

- This moderator action is unlikely to be done through the mobile user interface, since analysing the behaviour of the user requires concentration.
- A locked developer can still remove his widgets.
- Are the widgets of the developer locked automatically when the developer is locked? What if the widget has multiple developers, some of which are locked and some of which are not?

Moderator-004: Unlocking a widget

Preconditions

The moderator is examining a locked widget and thinks that the widget is not problematic after all. The moderator has locked the widget himself or has an administrator status.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The moderator clicks a button "Unlock this widget" and describes the reason for unlocking the widget.
2. The widget's page is restored. A notification is sent to the widget's developers.

Exceptional conditions

N/A

Notes

- The platform might check if some of the user's widgets are unlocked and inform the user.
- Are there some other reasons for locking than malicious behaviour? E.g., the widget is outdated, its developers are not active any more and the widget does not work.

Moderator-005: Unlocking a user

Preconditions

The moderator is examining a locked user and thinks that the user can be unlocked (either there was no proper reason for locking or the user has already been locked for a long enough time). The moderator has locked the user himself or has an administrator status.

Use cases before

User-009: Examining a user's profile (Section 2.1.2, page 14).

Use cases after

N/A

Normal progress

1. The moderator clicks a button “Unlock this user” and describes the reason for unlocking the user. A notification is sent to the user. Also the other moderators and administrators are notified.
2. The user’s page is restored and the user is able to participate in the system.

Exceptional conditions

N/A

Moderator-006: Removing feedback

Preconditions

A user has entered spam comments (e.g., advertisements or plain rubbish) or the user account has been captured and thus the feedback entered is not from the real user.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The moderator sees the feedback items (ratings and comments) given to the widget.
2. The moderator clicks a “Remove” button near the feedback item he wants to restore.
3. The feedback item is greyed out in the list and can only be seen by moderators. Other users do not see the feedback.
4. Report from this action is added to the moderator log.

Exceptional conditions

N/A

Moderator-007: Restoring feedback

Preconditions

A moderator has removed a feedback item but now it needs to be restored (e.g., the moderator removed the feedback item by accident).

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after
N/A
Normal progress

1. The moderator browses the feedback items of a widget.
2. The moderator clicks a "Restore" button near the removed feedback item he wants to restore.

Exceptional conditions
N/A

2.1.5 User cases for the administrator

Administrator-001: Granting moderator status to a user

Preconditions
The administrator is examining a user's profile and has decided that the user is reliable and wishes to grant moderator status to the user. This decision may be done because of, e.g., the user has participated in forum discussions, provided good quality widgets etc.
Use cases before
User-009: Examining a user's profile (Section 2.1.2, page 14).
Use cases after
N/A
Normal progress

1. The administrator clicks a button "Grant moderator status". A notification is sent to the user.

Exceptional conditions
N/A

Administrator-002: Removing moderator status to a user

Preconditions
The administrator is examining a user's profile.
Use cases before
User-009: Examining a user's profile (Section 2.1.2, page 14).
Use cases after
N/A

Normal progress

1. The administrator clicks a button "Remove moderator status". A notification is sent to the user.

Exceptional conditions

N/A

Administrator-003: Removing a widget permanently from the system

Preconditions

The administrator is examining a widget and thinks that the widget is dangerous or inappropriate, possibly after reading an abuse report on the widget. The widget has possibly been already locked. (See Notes for discussion about the reasons for removing a widget.)

Use cases before

Moderator-001: Reading an abuse report (Section 2.1.4, page 25) (not necessary), User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The administrator clicks a button "Remove this widget" and describes the reason for removing the widget.
2. The developers of the locked widget are notified (e.g., by sending an e-mail). Also the other administrators and moderators are notified (e.g., showing the locking in a log of recent moderator activity).
3. The widget can no longer be found from the system. If someone tries to access the widget via a direct link, an error message is shown.

Exceptional conditions

N/A

Notes

- The Widget Platform might check if some of the user's widgets are removed and inform the user.

- This administrator action is unlikely to be done through the mobile UI, since analysing a widget requires concentration.
- Is there some other reasons for removing than malicious behaviour? E.g., the widget is outdated, it's developers are not active anymore and the widget does not work. Is locking enough in those cases?

Administrator-004: Removing a user permanently from the system

Preconditions

The administrator is examining a user and thinks that the user is malicious or misbehaving.

Use cases before

User-009: Examining a user's profile (Section 2.1.2, page 14), also possibly: Moderator-001: Reading an abuse report (Section 2.1.4, page 25) and User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

1. The administrator clicks a button "Remove this user" and describes the reason for removing the user.
2. The user is removed from the system and a notification describing a reason for locking is sent to him. (The data is not deleted, though, so that the user can be restored.) Also the administrators, and other moderators are notified (e.g., showing the locking in a log of recent moderator activity).
3. If the user tries to log in (or is already logged in and tries to do something), the system displays a message saying the the user account has been deleted. If the removed user is a developer, his name may still be shown with the widgets? If the removed user is a developer, he may be allowed to remove his own widgets from the system? What about co-authored widgets?

Exceptional conditions

N/A

Notes

- This moderator action is unlikely to be done through the mobile user interface, since analysing the behaviour of the user requires concentration.

- This use case must be possible because it must be possible to remove user accounts created maliciously for "spamming the user name space".
- Are the widgets of the developer locked automatically when the developer is locked? What if the widget has multiple developers, some of which are locked and some of which are not?
- What are the rights of a removed developer? To e.g. remove their own widgets?

Administrator-005: Restoring a user to the system

Preconditions
The user has removed his user account and wants it to be restored.
Use cases before
User-002: Unregistering from the system (Section 2.1.2, page 8).
Use cases after
N/A
Normal progress

1. The administrator sees a list of removed users and searches the user account by starting to type the user name. The list also shows information about the users, e.g., when registered / when unregistered or removed and whether they unregistered themselves or were removed (and if removed, the reason for removal).
2. The correct user name is shown when the administrator has typed enough letters to identify the name.
3. The administrator clicks "Restore user account".
4. The system displays a message stating that the account was restored.

Exceptional conditions
N/A

2.2 Experimental use cases

The reputation mechanism of the Widget Sharing system affects the use cases of the system. This page describes use cases which may or may not be present in the Widget Sharing system. These use cases depend on, e.g., the reputation

mechanism or other aspects not yet agreed on. They also overlap partially with each other.

Use cases for the user:

- Declaring friends (Section 2.2.1, page 36);
- Recommending a widget to friends;
- Recommending a widget to a particular person / Inviting a person to use a widget (Section 2.2.1, page 36); and
- Giving detailed ratings (Section 2.2.1, page 37).

Use cases for the developer:

- "Supporting" widgets (Section 2.2.1, page 37); and
- Asking for assessment for an own widget so that it could be declared trustworthy.

Use cases for the moderator:

- Maintaining a list of interesting (trustworthy?) widgets ("The system's pick");
- Acknowledging trustworthy / good widgets (Section 2.2.1, page 38);
- Acknowledging trustworthy / good developers (Section 2.2.1, page 38); and
- Writing reviews. (Does someone have a possibility of commenting the reviews?)

Other:

- Stating that a widget is trustworthy (Section 2.2.1, page 39). (Required role for this use case?); and
- Stating that a user is trustworthy (Section 2.2.1, page 40). (The concept of "friends" overlaps with this.)

2.2.1 More detailed description of the experimental use cases

User-EXP: Declaring friends

Description

The user declares that another user is his friend.

Rationale

- The user might trust more the widgets which are used by his friends.
- The user might find interesting widgets by browsing the widgets downloaded / rated by friends.
- The user might trust more the widgets which are developed by his friends / his friends' friends.

Effects on other use cases

Information shown on a widget's page might include:

- list of friends who have downloaded / rated this widget;
- average rating given by the friends (in addition to the global average rating); and
- comments given by the friends might be highlighted.

Other information:

- the front page / notification page might show new widgets downloaded / rated by friends (the user might find interesting widgets that way); and
- the user's page might include a list of friends: this enables the user to browse the social network.

Notes

- How do people find friends?
  - Real life friends give their information (e.g., user name) and the user can search them in the system (User-013: Searching users (Section 2.1.2, page 18)).
  - Virtual friends?

User-EXP: Inviting a friend to use a widget

Description

The user invites a friend to use a widget. The friend then sees this invitation in a notification page or is notified by sending an e-mail.

Rationale

The user might want his friends to use the same application. E.g., a widget which shows which books the friends are reading.

Effects on other use cases

The invitations must be shown somehow. Notification system inside the Widget Sharing system?

User-EXP: Giving detailed ratings

Description

When the user rates a widget, he has the opportunity to rate several aspects of it. Some aspects could be related to the "normal" rating (how useful / entertaining the widget is) and some ratings could be related to the safety / security / trustworthiness of the widget.

Rationale

This way the Widget Sharing system could provide more information to the users and also gain more information, e.g, for assessing the trustworthiness of the widget.

Effects on other use cases

N/A

Notes

- Normal users are not capable of assessing the safety / security / trustworthiness of the widget. They should not be encouraged to rate these aspects.
  - Could rating safety / security / trustworthiness be available to only a subset of users (developers? moderators?).
- The aspects might include:
  - user interface / outlook of the widget, smoothness of usage, reliability (lack of errors), usefulness, how entertaining the widget is; and
  - trustworthiness (the widget is not malicious).

Developer-EXP: Supporting a widget

Description

The developer (user?) transfers part of his own reputation to a widget not developed by him, e.g., to a widget developed by his friends.

Rationale

Widgets recommended by famous users are probably more trustworthy.

Effects on other use cases

Information shown on a widget's page might include:

- names of recommenders.

Notes

- Who can recommend? Only developers / other users as well?
- What is the incentive for recommending widgets?
  - More reputation to oneself if the widget indeed is good?
  - Only the social aspects? ("If I recommend my friend's widgets, he recommends mine and thus my widgets get more users.")
- Could this be done automatically if the system knows who are friends?

Moderator-EXP: Acknowledging good widgets

Description

The moderator states that a widget is interesting (useful / entertaining) and good (not malicious).

Rationale

Users can trust widgets which are acknowledged more.

Effects on other use cases

- Information shown on a widget's page might include the acknowledgements.
- This information could also be made visible in widget lists (e.g., by adding golden frames to the widget icon).

Notes

See "Moderator-EXP: Stating that a widget is trustworthy" (Section 2.2.1, page 39).

Moderator-EXP: Acknowledging good developers

Description

The moderator states that a developer has done good work in the Widget Sharing system, e.g., by providing good widgets (or e.g. sending relevant abuse reports).

Rationale

- Users can trust widgets whose developers are are acknowledged more.
- Getting acknowledgement motivates the developers to continue providing good widgets.

Effects on other use cases

- Information shown on a widget's page and developer's page might include acknowledgements.
- This information could also be made visible in widget lists (e.g., by adding golden frames to the widget icon).

Moderator-EXP: Stating that a widget is trustworthy

Description

The user (developer? moderator?) states that a widget is trustworthy and a textual comment describing the reason for the decision:

- the widget's source code has been read;
- some tests have been run; or
- the widget's developers are trustworthy.

Rationale

The normal ratings describe how entertaining / useful the widget is and it's difficult to deduce how trustworthy the widget is. This would provide direct data for measuring the trustworthiness.

Effects on other use cases

Information shown on a widget's page might include:

- these statements of trustworthiness and their justification.

Notes

- Who is entitled to evaluate the trustworthiness of a widget?
- Are there people using the system who want to spend their time evaluating the trustworthiness?
- What is the incentive for doing so?

  - If there is an incentive: people might want to rate random good-looking widgets as trustworthy without reading the source code / without any other extra effort.
  - If most widgets are indeed trustworthy, it is difficult to find out who is cheating and who is really reading the source code.
- Widgets may change their source code.
- What about the opposite case: should it be possible to state that a widget is not trustworthy? (E.g., if the widget sends location information to a server and it is not clear what the server does.)

Moderator-EXP: Stating that a user is trustworthy

Description

The user (developer? moderator?) states that a user is trustworthy. This use case overlaps partially with the concept of "friends", e.g, we could assume that declaring trustworthiness is the same thing than declaring friends.

Rationale

- The normal ratings describe how entertaining / useful the widget is and it's difficult to deduce how trustworthy the widget developer is.
- It may be that the developer is a trustworthy person but has not provided useful / entertaining widgets yet.
- This would provide direct data for measuring the trustworthiness of the person.
- Ratings of trustworthy persons could have more weight.

Effects on other use cases

Information shown on a widget's page or on a user's page might include these statements.

Notes

- Who is entitled to evaluate the trustworthiness of a user?
  - Do the persons have to know each other in real life?
- What is the incentive for giving these statements?

  - If there is an incentive: people might want to rate random users as trustworthy without reading the source code / without any other extra effort.
  - If most users are indeed trustworthy, it is difficult to find out who is cheating and who is honest.
- Users may change their behaviour so that a person is first considered trustworthy but then starts to misbehave.
- What about the opposite case: should it be possible to state that a user is not trustworthy?

2.3 Threat analysis

We use a threat matrix approach: first we identify a set of actors who are the central sources and targets of threats, then produce a matrix with this set forming both the columns (threat causers) and rows (targets of threats). At each crossing, we identify what threats an actor of the type in the column can cause to the type of actor in the row. This is an application of the threat tree method [Amo94].

Threats are directed towards assets, which are abstract or concrete things of value to the target actor that can be harmed or destroyed. Threats are possibilities, and they can be assigned a severity (how bad it is if the threat becomes reality) and a probability (how likely it is that the threat becomes reality). A threat with a severity and a probability is also called a risk. (More in the Coras technical report [LdBSV04].)

Some threats are activated by attacks. An analysis of attacks can be used to determine what kinds of countermeasures should be taken to deter them, and behavioural patterns of malicious actors are commonly used in simulations to evaluate what kinds of attacks a system can handle [SVB06]. For examples on attacker patterns in reputation systems. Different attacks have different costs; it is usually not possible or worth the effort to defend against a highly resourceful and determined attacker, e.g. a government bent on subduing the system, while even a slight increase in the cost of performing a successful attack can effectively deter casual mischief or an attacker seeking profit. These aspects are considered further in the Misuse cases analysis in Section 2.4.

Note that well-behaved users and general non-attackers can also cause threats by just doing what they are supposed to; e.g. too many users can overload a popular server, or a moderator locking a malicious widget will cause the widget to not be used, which is not in the interest of the widget

developer.

2.3.1 Actors and their assets

We have identified eight actors:

- a widget,
- the rating and widget sharing system,
- a third party service provider (accessed by a widget),
- a widget user,
- a rater (user giving a rating); including possible expert raters,
- a ratee (target user/widget of rating),
- a widget developer, and
- a moderator (handler of security reports, not the owner of the entire system as such). In addition, the "Other" actor represents everything not categorized here. The owner of the entire system ("system administrator" in e.g. the use case analysis in Section 2.1) is not considered as a separate role in this threat analysis.

Some assets we have identified so far:

- Data of the widget user
  - given by the user to the widget sharing system;
  - given by the user to a widget s/he is running; also, implicitly, the user's interactions (e.g., news items read by the user);
  - stored on the user's device (e.g., an address book, browsing history);
  - available in a widget service / on a service site the widget accesses; and
  - invested by the user in a service accessed by the widget (i.e. bank account).
- Privacy
  - against eavesdropping (sound / microphone),
  - against location tracking (location / GPS),

  - against spying (images / camera),
  - against spying (video / camera), and
  - against sharing sensitive data (e.g. address book).
- Security (threat: unauthorized access)
  - access to a third-party service,
  - access to device resources, and
  - access to (sensitive) information.
- Resource availability / costs
  - battery life,
  - CPU availability,
  - network resources, and
  - "costly" communication resources (phone, SMS).
- Credibility of a rater.

Assets specific to widget developers:

- Reputation
  - personal reputation, and
  - "reputation" of own widgets.

Assets specific to the sharing system:

- general trust in widget sharing system,
- users of the system (user participation), and
- usability, easiness of participation (the users are not required to do too much work, e.g., provide ratings for ratings, to use the system).

Assets specific to the widget:

- Functioning as intended (i.e. we assume the widget "suffers" from not functioning as intended).
- Good reputation of the widget (if the widget is not malicious, or if it is).

Assets specific to the moderator:

- Not having too much work to do.
- Receiving valid reports.

2.3.2 The threat matrix

We examine threats caused by one type of actor to all other actor types, and the "Other" category for completeness.

Widgets causing threats

- To widgets:
  - The widget can modify/remove data used by other widgets or itself to function.
  - The widget can cause a denial of service to other widgets or itself by e.g. overusing the resources available on the device (hogging all the CPU/memory, reserving resources that cannot be shared — like taking a picture, "reserving" the speaker etc.).
- To rating system:
  - The widget can antagonize its user by misbehaving, and damage the system's overall "reputation".
  - Theoretically the widget can misbehave so badly the system owners get sued.
- To third-party service:
  - A widget can overload the service (DoS).
  - A widget can malfunction, do something the user would not do
    - repeat intended activity (multiple posts spam etc.);
    - not do intended activity (fail to complete a transaction, leaving resources reserved etc.); or
    - generate new activities on its own (send payments to third parties etc.).
- To widget user:
  - A widget can record audio/video/location without the user knowing (threat to device space, and basis for other threats).
  - A widget can pass audio/video/location information on without the user's consent.
  - A widget can pass on random information on the device (or prompted from the user) without the user's consent.

  - A widget can modify random information on the device without the user knowing/consenting.
  - A widget can delete random information from the device without the user knowing/consenting.
  - A widget can fail to pass on information intentionally provided by the user or available on the device for that purpose, or pass on incorrect information.
  - (A widget can read random information not intended for it without the user knowing.)
  - A widget can use up resources needed for other activities, such as processing power, battery life, memory and disk space, network capacity.
  - A widget can access pay-per-use services (i.e. make phone calls, send messages, access a third party web service with per use pricing) needlessly, causing costs to the user.
  - A widget may generally not work (do what it was supposed to)
    * a widget may not do what it was supposed to in a specific situation (e.g. not alert for help, fail to send message etc. — reliability threat); or
    * a widget may do something intended too many times, or not do it at all.
  - A widget can do something unexpected (elaborated in other threats).

- To rater: N/A
- To ratee: N/A
- To widget developer:
  - A widget may have a bug and behave in a way unintended by the developer, threatening his/her reputation.
- To moderator:
  - A malicious or malicious-looking widget can cause users to flood the moderator with security reports.
- To other: N/A

Rating system causing threats

- To widget:
  - A widget may become too popular or not popular enough, see threats caused by raters to widgets.
- To third-party service:
  - See threats caused by raters to widgets (too popular / not popular enough); applies to third party services that get bogged down or rely on many widget users.
- To itself: N/A
- To widget user:
  - See threats caused by single raters to the user (misleading through reputation); through an aggregated threat from the raters, or the rating system presenting the information in a misleading way.
- To rater:
  - (The system threatens the rater's privacy by showing their ratings/reports, which can lead to antagonizing other users or a loss of credibility if the rater is considered to be wrong.)
- To ratee:
  - See threats caused by raters to ratees. The system can present the ratee's reputation as bad, deserved or not, or lose positive reputation data.
- To widget developer:
  - See threats caused by raters to widget developers. The system can present a widget developer's reputation as bad, deserved or not.
  - The system can lose the developer's widget. (Delete by accident.)
- To moderator:
  - See threats caused by raters to the moderator. The rating system can overwork the moderators through reports.
- To other: N/A

Third party service causing threats

Most threats that can be caused by a widget can also be done directly or indirectly by the third party service accessed by the widget. To avoid repetition, we consider these to be threats caused by the widget which has been set to access a malicious service, or to use a benign service maliciously. The widget developer may be convinced that the third party service is benign, while e.g. a cross-site scripting (XSS) vulnerability or some other security hole in the third party service can cause the widget to behave unexpectedly. The third party service chosen by the developer can also be a malicious service masquerading as a benign one (such as a phishing site that looks like a popular net bank).

The third party service can easily influence those parts of a widget's behaviour that depend on the service itself, and can relatively easily use up unnecessary amounts of processing power and memory on the device. However, for the service to make the widget access protected APIs on the device (such as taking pictures), the widget must either be running code it receives from the third party service, or be designed in a way that input from a third party service triggers protected activities. In the cross-site scripting attack, HTML content intended to be shown to the user contains scripting code inserted by the attacker, which then gets executed by the browser — or the widget, in this case.

- To widget:
  - The service can either intentionally or unintentionally make the widget malfunction, which activates threats caused by the widget to others.

Widget user causing threats

- To widget:
  - The user can use the widget in an unexpected way, causing it to malfunction.
  - See threats caused by raters: too popular widgets suffer from too many widget users.
- To third-party service:
  - A malicious (or strange) user uses the widget in an unexpected way to attack the third-party service, spam it etc.

- To rating system:
  - Too many users can overload the rating/sharing system. Too few can make it unsuccessful.
- To widget user:
  - The widget user can harm himself in multiple ways, from downloading malicious widgets to using them in detrimental ways.
  - Users influencing each other through ratings: see rater causing threats to widget user.
  - A user of a collaborative widget can attack another user via the widget, through the collaboration aspect (e.g. playing poker through widgets) or through a security hole opened by the widget (turning the widget itself into a "trojan horse" or a backdoor).
- To rater:
  - Widget users influence raters mainly as raters themselves, see threats caused by raters to raters.
- To ratee:
  - See threats caused by raters to ratees. A widget user can also misunderstand how a widget works and provide undeservedly bad ratings as a result.
- To widget developer:
  - See threats caused by raters to widget developers.
- To moderator:
  - The widget user attacking other users can lead to more security reports to the moderator as well, even though the widget itself is not malicious, just insecure.
  - See also threats caused by raters to moderators.
- To other: N/A

Raters causing threats

- To widget:
  - Positive ratings can cause widget to become so popular its third-party service becomes overloaded.
  - A widget that depends on having a large user base may also not function as a result of not being popular enough. Otherwise it is not considered a problem of the widget if no one uses it — it still functions as it should.
- To third-party service:
  - See threats by raters to widgets.
- To (rating) system:
  - Can depress users (targets of ratings) and drive them away.
  - Can adversely affect the system's "reputation" / overall credibility by malicious/low quality ratings.
  - Can be inactive, not participating in the system.
  - Can twist ratings in general; too negative or too positive (basis for an attack).
- To widget user:
  - Can mislead widget user to use a bad widget by good ratings, or by failing to report it.
  - Can mislead widget user to not use a good widget.
  - See widget: too popular widgets (rater causing threat to widget).
  - Can discourage widget developers from submitting widgets (of a given type, or in general) by rating them too low.
- To raters:
  - Can reduce credibility of other raters or self through disagreement.
- To ratee
  - Can give unfair/fair negative ratings to other users/widgets.
- To widget developer

  - Can cause a single widget to have low reputation and not get users.
  - Can cause a developer to have low reputation, causing all of his/her widgets to not be used.

- To moderator:
  - Can overload moderator with valid/invalid security reports.

- To other: N/A

Ratee causing threats

The ratee is a target role; the widget developer causes threats when not in the ratee role as such.

Widget developer causing threats

- To widgets:
  - The developer can develop a buggy widget, causing the widget to not work as intended.
  - The developer can create widgets causing threats to widgets, see above.

- To third-party service:
  - See widgets causing threats to third-party service. The developer can also consciously direct users to access a third-party service in unintended ways.

- To rating system:
  - The developer can increase the ratio of negative ratings through developing bad widgets, leading to a reduction in the perceived quality of widgets shared in the system.
  - The developer can use up resources in the sharing system for unintended purposes through misusing the ability to store widgets on the server.

- To widget user:
  - Can develop a widget which misbehaves (see widget to user threats).

- To rater:

  - Can reduce rater credibility by modifying widget on the fly after receiving ratings.

- To ratee:
  - Can cause other developers to receive bad ratings through introducing bugs/malicious code into a jointly-developed widget, or by producing vastly superior widgets that cause users to become more demanding.

- To widget developer:
  - Can hog all the users of a specific type of widget (or hijack a widget as his own), causing a loss of reputation and notoriety for them.

- To moderator:
  - Can produce widgets that cause a flood of security reports, either correct or incorrect.

- To other: N/A

Moderator causing threats

- To widgets: See moderator causing threats to widget developer.

- To third-party service:
  - Can cause third-party service to not be used by blocking a widget generating traffic to it; or
  - Can allow a third-party service to be misused by not blocking a widget designed to misuse it.

- To rating system:
  - Can damage the general trustworthiness of the system either by being too eager to remove widgets, or too slow to react to security reports.

- To widget user:
  - Can remove a useful/desired widget; or
  - Can fail to remove a malicious widget.

- To rater:

  - Can cause the rater frustration by not reacting to reports; or
  - Can punish a rater for warnings deemed undeserved.

- To ratee:
  - Can remove a widget, causing a loss of gathered positive ratings, or otherwise punish the developer via their reputation.

- To widget developer (see also moderator causing threats to ratee):
  - Can punish the developer through means other than their reputation, e.g. blocking access to their account.

- To moderator:
  - Can inadvertently encourage users to submit more reports than should, or to not submit reports that should be.

- To other: N/A

Other actors causing threats

One typical threat caused by other actors is an outsider impersonating any existing user role. This allows the outsider to both attack other users, and to direct blame and various negative consequences to the user who it is impersonating. We consider the threat of attacks to be caused by the role being impersonated, while the latter type of consequence threat is specifically a threat caused by outsiders, "other actors".

The second typical threat caused by outsiders are automated attacks which may not be aimed particularly at a given system. For example, spambots can forage the Internet for instances of a given forum engine or open instances of a popular wiki engine, and spread spam or deface pages without being aware of what kind of service the target is providing.

- To widgets:
  - Infrastructure builders can cause widgets to stop working e.g. by updating platform interfaces.

- To third-party service:
  - An outside attacker can bring down the third-party service, causing loss of data or denial of access.

- To rating system: N/A (Barring the usual random denial of service attacks, etc.)
- To widget user:
  - Similarly to other widget users causing threats to a widget user, an outside attacker who is not a widget user as such can use a security hole opened by a widget to attack the widget user.
  - An outsider can impersonate a widget user and remove their account.
- To rater: An outsider can impersonate a rater to reduce their credibility.
- To ratee: An outsider can impersonate a developer and modify their widget to attract bad ratings.
- To widget developer:
  - An outside attacker can bring down the sharing server, causing loss of widgets and data.
  - An outside attacker can bring down the third-party service the developer's widget relies on, causing widgets to not be used.
  - An outsider can impersonate a developer and remove their account.
- To moderator: An outsider can bot-spam through reports.
- To other: N/A

2.4 Misuse cases

Terminology:

asset something good we want to protect,

threat something bad happening to an asset,

risk threat + its probability + its impact,

attack activating a threat intentionally.

Misuse cases are divided into two groups:

- Misuse cases by the widget: misbehaviour carried out by the widget

  - A widget may misbehave because it is designed to do so or because it is under control of a malicious third party (e.g., the server used by the widget has been attacked or the widget has a security hole).
  - Multiple widgets may misbehave together (e.g., one widget reads the user's data and saves it into a file and another widget reads it from the file and sends it to a server).
  - A widget may interact with third parties (computer systems or people) as a part of the misuse case.
  - Sometimes it is OK for a widget to send data to a server but not OK for the server to store it. For example, a widget for finding nearby restaurants may send location data to a server. By examining the widget it is impossible to know if the server does malicious things with the data (e.g., stores it against the user's will). The widget and the server may be programmed by different people.

- Misuse cases by the users of the Widget Sharing system:
  - malicious / unwanted behaviour in the Widget Sharing system.

2.4.1 Misuse cases by the widgets

The list of misuse cases by the widget is based on the list of widget capabilities in Appendix B.

Here, "user's private data" refers to data stored on the user's device or available to the widget, such as:

- phone book,
- data stored by other applications, e.g., browsing history,
- the user's files (e.g., pictures, text documents),
- data that can be acquired by the widget (e.g., still images, video, audio, location data), and
- data describing the interaction between the user and the widget (e.g., if the widget is a news client, the interaction data includes information about what kind of news the user has read).

Network/cost-related groups

- Phone calls:
  - Making an unwanted phone call and thus producing costs;
  - Using phone calls for eavesdropping; and
  - Reserving the resource so that other applications cannot use it (e.g., no incoming phone calls can be answered during a phone call).
  - See also: multimedia recording.
- Call Control (call set-up or tear-down):
  - See above.
- Net Access (network data connection, e.g., GSM, GPRS, UMTS, mostly http usage):
  - Note: the widgets could be restricted so that they can only connect to certain hosts specified in the manifest. How does this limit the threats?
  - Sending the user's private data to a server (if the data is not gathered by the widget, the widget needs also access to read the data) without / against the user's will;
  - Phishing, e.g., passwords or credit card numbers;
  - Misbehaving in a third party service (e.g., misbehaving at a discussion forum by sending spam); and
  - Using the connection so extensively that other applications cannot use it (or can use it only to a small extent).
- Low Level Net Access (low level network data connection, e.g., Sockets):
  - Sending corrupted packets; and
  - Impersonation on a network level (claiming that messages come from IP address X).
  - See: Net Access.
- Local Connectivity (using a local port for further connection, e.g., COMM port, IrDA, Bluetooth):
  - Reading data from Bluetooth devices (e.g., accelerometer, heart rate sensor) and sending it forward.

  - See: Net Access.

- Messaging (sending or receiving messages, e.g, SMS, MMS):
  - Sending an unwanted message and thus producing costs; and
  - Sending spam (to numbers in the phone book or people in general).
- Restricted Messaging (something not normally available to the user, e.g., Cell Broadcast).
- Application Auto Invocation (starting other applications, e.g., timed execution):
  - Jamming the user's system by starting too many applications (also causing the jamming occur later by using timed execution); and
  - Causing the widget to start automatically again if it is terminated (is this possible?)
- Authentication (e.g., using an authentication method available in the phone, e.g., a password stored on the device):
  - Using the authentication to log in a system (and appear as if the user had logged in).

User-privacy-related groups

- Multimedia recording (any kind of multimedia, e.g., still images, video and audio):
  - Gathering data without / against the user's will; and
  - Sending the data to a server / to a third party (requires also network access) without / against the user's will.
- Read User Data Access (read any files, e.g., phone book).
- Write User Data Access (write / add / delete any files, e.g., phone book):
  - Modifying / removing the user's data;
  - Modifying / removing files which are necessary for other widgets; and
  - Creating files which take up the space on the device

- Smart Card Communication.
- Location (e.g., GPS, cell information):
  - Gathering data without / against the user's will; and
  - Sending the data to a server / to a third party (requires also network access) without / against the user's will.
- Landmark (targets marked by the user in a navigation software):
  - Sending the data to a server / to a third party (requires also network access) without / against the user's will.

Call Control, Low Level Net Access, Restricted Messaging and Smart Card Communication seem to be relatively uninteresting access rights for being used in widgets.

Note: widgets could "own" their files so that they can write / modify their own files but not any other files.

2.4.2 Misuse cases by the users of the Widget Sharing System

The structure for describing misuse cases by the users is as follows:
Actor-Number: Misuse case name

Misuser profile
: Assumptions about the misuser, e.g., how technically skilled the misuser must be and what is the incentive for this kind of misbehaviour.

Preconditions
: Conditions necessary to carry on this misuse case

Use cases before
: Use cases / misuse cases to be performed before this use case

Use cases after
: Possible use cases / misuse cases which can be performed after this use case

Normal progress
: The steps taken by the user to go trough the misuse case

Threats
: Threats activated by this misuse case. Also: use cases threatened by this misuse case.

Mitigation
: How this misuse case could be prevented / its effects mitigated. Also: use cases mitigating this misuse case.

Use cases for the user:

- Unfair ratings and comments:
  - User-M001: Unfair positive (Section 2.4.2, page 59) (e.g., ballot stuffing);
  - User-M002: Unfair negative (Section 2.4.2, page 60) (e.g., blackmailing);
  - User-M003: Spamming (Section 2.4.2, page 61) (a large amount of groundless ratings and comments); and
  - User-M004: Giving no feedback (Section 2.4.2, page 63).
- User-M005: Sending false abuse reports (Section 2.4.2, page 64);
- User-M006: Sending spam abuse reports (Section 2.4.2, page 65) (a large amount of abuse reports);
- Creating fake identities;
- Discarding an identity and starting anew; and
- Stealing an identity.

Use cases for the developer:

- Providing malicious widgets:
  - Developer-M001: Adding a malicious widget (Section 2.4.2, page 66);
  - Developer-M002: Modifying a widget to be malicious (Section 2.4.2, page 67);
  - Developer-M003: Modifying malicious widget to normal (Section 2.4.2, page 69); and
  - Types of malicious widgets: see above.
- Entering false co-developer information
  - Not entering a co-developer; and
  - Entering a fake co-developer.
- Modifying the widget without the co-developers' will:
  - Updating the widget source code;
  - Removing the widget; or

  - Modifying the widget's information (e.g., adding / removing co-developers).

Note: Can we assume that the developers to settle their disagreements outside the Widget Sharing system so that we can assume that developers do not do things against each others' will? (Or that people develop applications only with people they consider trustworthy so that these problems will not be present.)

Use cases for the moderator:

- Not handling an abuse report properly;
  - Categorizing a correct abuse report as spam or false alarm; and
  - Taking actions because of a false abuse report (spam or false alarm):
- Locking a widget which is not malicious; and
- Locking a user who has not misbehaved.

Misuse cases for the user

User-M001: Unfair positive ratings / comments

Misuser profile

- No technical skills needed.
- The user may be a fake identity created for ballot stuffing.
- Incentive: e.g., widget developers are the user's friends (is it a problem?)

Preconditions

The user is viewing a widget's page.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after

N/A

Normal progress

Same as in use case User-011: Rating a widget (Section 2.1.2, page 16).

Threats

- Information displayed in use cases User-008: Examining a widget (Section 2.1.2, page 12) and User-009: Examining a user's profile (Section 2.1.2, page 14) may be false:
  - The rated widget looks more entertaining / useful than it really is;
  - Its developers look better (more famous / more trustworthy) than they really are; or
  - Their widgets are possibly downloaded more and the developers' ratings are trusted more.

Mitigation

- The reputation mechanism should be resistant to ballot stuffing.
  - Recognizing ballot stuffing?
  - Mitigating the effects of ballot stuffing, e.g, weighting the ratings given by the users by taking the amount of contribution (e.g., developed widgets) into account.
- Locking / removing fake identities: Moderator-003: Locking a user (Section 2.1.4, page 27) and Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33).
- If the user is not a fake identity, can there ever be enough information to say that the user has entered too positive ratings / comments and lock / remove the user?

User-M002: Unfair negative ratings / comments

Misuser profile

- No technical skills needed.
- The user may be a fake identity created for blackmailing.
- Incentive: e.g., widget developers are the user's fiends / competitors (if the user is also a developer).

Preconditions

The user is viewing a widget's page.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after

N/A

Normal progress

Same as in use case User-011: Rating a widget (Section 2.1.2, page 16).

Threats

- Information displayed in use cases User-008: Examining a widget (Section 2.1.2, page 12) and User-009: Examining a user's profile (Section 2.1.2, page 14) may be false:
  - The rated widget looks less entertaining / useful than it really is;
  - Its developers look worse (less famous / less trustworthy) than they really are; or
  - Their widgets are possibly downloaded less and the developers' ratings are trusted less.
- Possibly the developers do not want to contribute to the Widget Sharing system if they feel like they have been treated in an unfair way.
- The Widget Sharing system does not look appealing.

Mitigation

- The reputation mechanism should be resistant to blackmailing with fake identities.
  - Recognizing blackmailing with fake identities?
  - For developers: a possibility to defend themselves against groundless negative comments?
  - Mitigating the effects of blackmailing with fake identities, e.g, weighting the ratings given by the users by taking the amount of contribution (e.g., developed widgets) into account.
- Locking / removing fake identities / misbehaving users: Moderator-003: Locking a user (Section 2.1.4, page 27) and Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33).

User-M003: Spam ratings and comments

Misuser profile

- No technical skills needed.
- The user may be a fake identity.
- Incentive: to harm the system, to advertise.

Preconditions

The user is viewing a widget's page.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after

N/A

Normal progress

Same as in use case User-011: Rating a widget (Section 2.1.2, page 16).

Threats

- Information displayed in use cases User-008: Examining a widget (Section 2.1.2, page 12) and User-009: Examining a user's profile (Section 2.1.2, page 14) may be false:
  - See User-M001: Unfair positive (Section 2.4.2, page 59) and User-M002: Unfair negative (Section 2.4.2, page 60);
  - Good-quality and bad-quality widgets look the same because a large amount of ratings is spam;
  - The same goes for developers.
- The Widget Sharing system does not look appealing.

Mitigation

- The reputation mechanism should be resistant to spamming with fake identities.
  - Recognizing spamming with fake identities?
- Locking / removing spamming users / fake identities: Moderator-003: Locking a user (Section 2.1.4, page 27) and Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33).

Notes

- It might be difficult to distinguish between spam and honest ratings. Some aspects:

  - If there are too many ratings per day, they are likely to be spam; and
  - On the other hand, if the user has downloaded many widgets before registering into the system, then registers into the system and rates all the downloaded widgets at the same time (of course, without downloading them again).

User-M004: Giving no feedback

Misuser profile

- No technical skills needed.
- The user may be a fake identity.
- Incentive: to harm the system.

Preconditions

The user is viewing a widget's page.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after

N/A

Normal progress

The user never executes the use case User-011: Rating a widget (Section 2.1.2, page 16).

Threats

- Information displayed in use cases User-008: Examining a widget (Section 2.1.2, page 12) and User-009: Examining a user's profile (Section 2.1.2, page 14) may be false:
  - No enough honest feedback to compensate the false feedback, see User-M001: Unfair positive (Section 2.4.2, page 59) and User-M002: Unfair negative (Section 2.4.2, page 60);
  - Good-quality and bad-quality widgets look the same because of the lack of ratings;
  - The same goes for developers.
- The Widget Sharing system does not look appealing.

Mitigation

- The Widget Sharing system should encourage giving honest feedback:
  - Simple interface, no much extra work; and
  - Benefits for users who give feedback? (Without encouraging spamming!)

User-M005: Sending false abuse reports

Misuser profile

- No technical skills needed.
- The user may be a fake identity created for blackmailing.
- Incentive: e.g., widget developers are the user's fiends / competitors (if the user is also a developer).

Preconditions
The user is viewing a widget's page.

Use cases before
User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after
N/A

Normal progress
Same as in use case User-012: Sending an abuse report (Section 2.1.2, page 17).

Threats

- A moderator may be mistaken by the abuse report and
  - Lock the widget Moderator-002: Locking a widget (Section 2.1.4, page 27); and
  - Lock the developers Moderator-003: Locking a user (Section 2.1.4, page 27).
- More work to the moderators, they have to read the abuse reports (Moderator-001: Reading an abuse report (Section 2.1.4, page 25)).
- Real abuse reports get less attention.

Mitigation

- Locking / removing the user sending false abuse reports Moderator-003: Locking a user (Section 2.1.4, page 27) and Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33).

Notes

- It might be difficult to distinguish false abuse reports from false alarms.

User-M006: Sending spam abuse reports

Misuser profile

- No technical skills needed.
- The user may be a fake identity created for blackmailing.
- Incentive: e.g., widget developers are the user's fiends / competitors (if the user is also a developer).

Preconditions

The user is viewing a widget's page.

Use cases before

User-008: Examining a widget (Section 2.1.2, page 12). Possibly: User-010: Downloading a widget (Section 2.1.2, page 16), User-014: Using a widget (Section 2.1.2, page 19).

Use cases after

N/A

Normal progress

Same as in use case User-012: Sending an abuse report (Section 2.1.2, page 17).

Threats

- A moderator may be mistaken by the abuse report and
  - Lock the widget Moderator-002: Locking a widget (Section 2.1.4, page 27); and
  - Lock the developers Moderator-003: Locking a user (Section 2.1.4, page 27).
- More work to the moderators, they have to read the abuse reports (Moderator-001: Reading an abuse report (Section 2.1.4, page 25)).
- Real abuse reports get less attention.

Mitigation

- Locking / removing the user sending spam abuse reports Moderator-003: Locking a user (Section 2.1.4, page 27) and Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33).

Notes

- It might be difficult to distinguish false abuse reports from spam:
  - If there are too many abuse reports per day, they are likely to be spam; and
  - On the other hand, if the user is, e.g., checking if a specific security hole occurs in widgets in the Widget Sharing system, then it is possible to enter many correct abuse reports. Difficult to distinguish even by examining the textual content of the abuse report (all abuse reports may have the same text describing the security hole).

Misuse cases for the developer

Developer-M001: Adding a malicious widget

Misuser profile

- Widget programming skills needed.
- Incentive: various incentives depending on the type of misbehaviour.

Preconditions
N/A
Use cases before
N/A
Use cases after
N/A
Normal progress

- The developer or developers develop a malicious widget.
- They upload it as in use case Developer-001: Uploading a new widget (Section 2.1.3, page 20).

Threats

- Specific to the malicious widgets, see above.

Mitigation

- Information shown on the widget's page (see User-008: Examining a widget (Section 2.1.2, page 12)) might indicate that the widget should not be trusted.
- Information shown on the developers' pages (see User-009: Examining a user's profile (Section 2.1.2, page 14)) might indicate that the developers should not be trusted.
- Users can send abuse reports (see User-012: Sending an abuse report (Section 2.1.2, page 17)) if they find out that the widget is malicious.
- The widgets can be locked (see Moderator-002: Locking a widget (Section 2.1.4, page 27)).
- The developers can be locked (see Moderator-003: Locking a user (Section 2.1.4, page 27)).
- The widgets can be removed (see Administrator-003: Removing a widget permanently from the system (Section 2.1.5, page 32)).
- The developers can be removed (see Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33)).

Notes

- Do we need other means for mitigation?
- How could it be made easier to find out that the widget is misbehaving?

Developer-M002: Modifying a widget to be malicious

Misuser profile

- Widget programming skills needed.
- Incentive: various incentives depending on the type of misbehaviour.

Preconditions

One of the developers has uploaded a widget and now a developer (not necessarily the same one) is viewing the widget's page.

Use cases before

Developer-001: Uploading a new widget (Section 2.1.3, page 20), User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

- The developer or developers modify an existing widget to misbehave.
- They update the widget as in use case Developer-002: Updating an existing widget (Section 2.1.3, page 21).

Alternative progress:

- The developer or developers program a widget which can download a source code for itself from a server.
- Then they change the source code so that a widget becomes malicious.

Threats

- Specific to the malicious widgets, see above.

Mitigation

- Information shown on the widget's page (see User-008: Examining a widget (Section 2.1.2, page 12)) might indicate that the widget should not be trusted.
- Information shown on the developers' pages (see User-009: Examining a user's profile (Section 2.1.2, page 14)) might indicate that the developers should not be trusted.
- Users can send abuse reports (see User-012: Sending an abuse report (Section 2.1.2, page 17)) if they find out that the widget is malicious.
- The widgets can be locked (see Moderator-002: Locking a widget (Section 2.1.4, page 27)).
- The developers can be locked (see Moderator-003: Locking a user (Section 2.1.4, page 27)).

- The widgets can be removed (see Administrator-003: Removing a widget permanently from the system (Section 2.1.5, page 32)).
- The developers can be removed (see Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33)).

Notes

- How is it possible to know that the widget has changed its behaviour and is malicious?
- If the rating mechanism reduces the reputation of a widget when it is updated (e.g., given ratings are not considered to apply wholly to the updated version).
  - Then there is an incentive to program the widget so that it can download a new source code for itself.
  - And that is probably not something what we want: Extra work for the developers, less transparent to users, no benefits for anyone.
- So, the rating mechanism should not punish developers for updating their widgets.

Developer-M003: Modifying a malicious widget to be normal

Misuser profile

- Widget programming skills needed.
- Incentive: various incentives depending on the type of misbehaviour.
  - In addition: counter-attacking the mitigation methods for misuse cases Developer-M001: Adding a malicious widget (Section 2.4.2, page 66) and Developer-M002: Modifying a widget to be malicious (Section 2.4.2, page 67) to be able to continue malicious behaviour longer.

Preconditions

One of the developers has uploaded a widget and now a developer (not necessarily the same one) is viewing the widget's page.

Use cases before

Developer-001: Uploading a new widget (Section 2.1.3, page 20), User-008: Examining a widget (Section 2.1.2, page 12).

Use cases after

N/A

Normal progress

- The developer or developers modify an existing malicious widget to behave correctly.
- They update the widget as in use case Developer-002: Updating an existing widget (Section 2.1.3, page 21).

Alternative progress:

- The developer or developers program a widget which can download a source code for itself from a server.
- Then they change the source code so that a malicious widget becomes normal.

Threats

- Specific to the malicious widgets, see above.
- The malicious behaviour of the widget is not noticed by Widget Sharing system moderators / administrators.
- A correct abuse report may be labelled as false alarm or spam.
  - Consequences to the user(s) who have sent abuse reports.

Mitigation

- Information shown on the widget's page (see User-008: Examining a widget (Section 2.1.2, page 12)) might indicate that the widget should not be trusted.
- Information shown on the developers' pages (see User-009: Examining a user's profile (Section 2.1.2, page 14)) might indicate that the developers should not be trusted.
- Users can send abuse reports (see User-012: Sending an abuse report (Section 2.1.2, page 17)) if they find out that the widget is malicious.
- The widgets can be locked (see Moderator-002: Locking a widget (Section 2.1.4, page 27)).
- The developers can be locked (see Moderator-003: Locking a user (Section 2.1.4, page 27)).

- The widgets can be removed (see Administrator-003: Removing a widget permanently from the system (Section 2.1.5, page 32)).
- The developers can be removed (see Administrator-004: Removing a user permanently from the system (Section 2.1.5, page 33)).

Notes

- While correcting a widget's behaviour is definitely beneficial, the misuse in this case lies in changing the widget's behaviour for the purpose of causing harm to abuse reporters of misleading a moderator.
- How is it possible to know that the widget has changed its behaviour and has been malicious but is normal again?

2.5 Discussion

This section presents some higher-level discussion on the analyses made.

2.5.1 Attracting useful and honest ratings

A central challenge in all systems involving collaborative filtering or reputation is how to attract useful and honest ratings from the users. Users benefit from the ratings of others through the new information the provide, while their own ratings seldom directly benefit them.

There are three sub-goals to attracting good ratings:

- Providing incentive for the user to give a rating in the first place (minimal requirement);
- Making the rating process easy and a part of the regular use, so that the threshold for rating is as low as possible (quantity of ratings); and
- Encouraging the user to extend effort into making the rating as high quality as possible (quality of ratings).

Beyond actually forcing users to give ratings, incentive may be the hardest to influence with technology alone. Users need to feel somewhat committed to the system storing their ratings in order to want to extend the effort to provide a rating. On the other hand, people are socially inclined to be helpful towards other people who they can identify with; faceless systems or organizations may not be so eagerly supported. The incentive problem also involves motivation

to provide information even when it does not require the user's effort at all: if the user feels that the information gathering which supports the rating system (such as lists of downloads, lists of widgets in use, etc.) invades on their privacy, they may be discouraged from using widgets in the first place. For example the information that a widget is being used actively by others could be taken into account when assessing its trustworthiness; for this purpose, users will have to be convinced that collecting the information needed for generating statistics like this is in their interests.

Making the rating process easy and a natural part of the user experience enters the field of user interface design. Minimizing the effort required for giving feedback is central; for example having to open a browser and to enter a website separately just to give a rating can make the user redecide on wanting to contribute. Bringing the rating into a part of the use of the widget without interfering on the user's goals is a challenge, as is making logging in easier for the user without opening them for new threats or invading on their privacy.

Encouraging the users to make high quality ratings once they have already decided to give a rating is a goal slightly contradictory to attracting as high quantities of feedback as possible. For example, allowing anonymous ratings can increase the sheer volume of ratings, but the trade-off of receiving more low-quality ratings such as unfounded claims or completely unrelated spam makes the option unattractive. The border between rating and voting is flexible: in both cases, the fairness principle of e.g. one say per user is desirable, but in addition, rating can either be a relatively private activity, only connected to a specific widget, or it can be a social activity, where the rating becomes a part of the identity of the user and e.g. shown in his/her profile.

2.5.2 Empowering users to control their risk through widget capabilities

Managing users' trust in widgets is a question of evaluating and controlling risk. Most of the risks involved come from the capabilities of widgets to harm the user in various ways, ranging from frustration to considerable monetary loss.

Few users acquire new devices and software in order to perform exhaustive risk analysis on them; the task is considered to go beyond the functional goals of the user, and is therefore easily neglected. In addition, not many users have the experience needed to understand how a widget and the software and hardware below it work; they just use it for a task, and are generally uninterested to investigate further as long as they achieve their functional

goals set for the widget.

Empowering users to control their risk therefore involves two separate goals:

- Helping the user understand the risks being controlled; and
- Providing an intuitive interface to managing the widget capabilities in order to control the risks.

Helping the user to understand risks involves visualizing the widget's requested capabilities to the user in an interesting, informative and understandable way, and communicating what kind of risk trade-off the user is making to gain the new functionality.

Providing an intuitive interface to managing the widget capabilities allows the user to control the risk conveniently after having made an informed decision about what kinds of trade-offs are desired. The manifest system being discussed forces the widget to express what kinds of access to protected APIs, such as making a call or connecting to the Internet, the widget needs in order to operate. In addition, the protected APIs may allow more fine-grained control of the widget's behaviour as well: instead of allowing photos to be taken by a widget at any time, it may be forced to request permission to take a picture each time. While for many types of functionality constant queries are unwieldy and frustrating to the user, a widget which needs to take one or two pictures during its entire lifetime can also do without an eternal permission to take pictures whenever it wants.

Further refinement to controlling widget capabilities can be gained by specifying certain allowed parameters for high-level activities as well, such as

- for an Internet connection, specifying which hosts (or even URLs) the widget is allowed to contact (and similar restrictions to e.g. phone calls, messaging or connection to external devices); and
- for a file system read/write access, specifying what files or directories the widget is allowed to access and how much space it is allowed to occupy.

There is, however, an important distinction between the user interface for controlling the capabilities and the capabilities possible to control: even though a user might simply move a paradigmal "paranoia level adjuster" to reflect their general attitude towards risk, the platform should be able to translate this into sensible risk management policies.

For a more detailed list of widget capabilities, see Section 2.4.1 and Appendix B.

Chapter 3

Final concept model

This chapter describes the emerging trust model for our proposal for a trust-promoting reputation mechanism for widget sharing. The trust model consists of several parts, including a description of our philosophical assumptions and concept definitions, together with a theoretical description of how the reputation mechanisms should operate. A separate section deals with risk management: how the system empowers the user to make educated decisions based on updated risk information. The handling also includes an analysis of risk tolerance and its effects on the emerging model. The role of moderator in the model, as well as how abuse reports are handled and how they affect the system, are described in their own subsection.

The validation section compares the designed system to the incentives presented in Section 3.1 and misuse cases described in Section 2.4. Further validation and experimentation are discussed in Chapter 4.

The chapter consists of five sections:

3.1 Philosophy — definitions, design principles, user incentives, data sources

3.2 Reputation mechanisms

3.3 Risk management

3.4 Moderators and abuse reports

3.5 Validation

The trust model is for a reputation mechanism proposal that is fundamentally built on the assumptions of a decentralized, community-based, open system where anyone can contribute and participate on free will and free of charge. The system is assumed to be self-sufficient and self-organizing in

the sense that minimal (if any) administration besides the various administrative powers of different type of users (on scale of e.g. novices/established users/developers/moderators etc.) is assumed. Users can join and withdraw from the system at any time.

The trust towards and trustworthiness of the widgets is based on that of the users of the system, especially on their reputation that is built over time based on user actions and feedback from other users on their actions and contributions. In order to create a rich base for trust-decisions, feedback mechanism with easy access are included wherever possible. The trust-effecting contributions can consist of various ingredients, including one or more of the following: widget contributing; rating of the existing widgets including abuse reporting; acting as moderator; writing reviews; contributing in discussion forums. Benevolent and useful contributions as judged by other users of the system will cause the reputation, and therefore, perceived trustworthiness, of the user to grow. The gained level of reputation diminishes if the user behaves maliciously. The initial level of reputation of a novice user is not zero but can also decrease from its initial value due to bad behaviour. Misbehaving users can also be frozen as punishment by moderator decision to indicate they are not trustworthy: Users who have completely lost their trustworthiness can be banned from the system. A widget's trustworthiness or lack of it can be demonstrated by abuse reports and, finally, freezing and/or removing the widget from the system by a moderator. The reputation — and thus, trust — is assumed to be built slowly over time, with small increments from the initial value.

Relative simplicity and ease of use have served as key design drivers in how the reputation mechanisms are intended to work, as users looking for widgets will generally not be willing to spend considerable amounts of time learning how to use the system.

3.1 Background

This section presents and discusses the basic definitions and assumptions behind our approach to the reputation and risk management mechanisms for widget sharing. It is divided into four subsections:

3.1.1 Definitions

3.1.2 Assumptions

3.1.3 Incentives for the users

3.1.4 Data sources available for the reputation/risk mechanisms

3.1.1 Definitions

- *Trust* is defined as a widget user's willingness to install and use a widget, considering the risks and incentives involved [RK05]. A *trust decision* is made by a widget user using information provided by the widget sharing system (the widget's overall trustworthiness assessment) on
  - the *reputation* of the developer,
  - the *risk* involved in installing/using it, and
  - the *risk tolerance* of the user.

Trust is subjective; reputation and risk are objective (i.e. global), while risk tolerance is subjective.

- The reputation of a developer measures the good/bad *ratings* and *verified abuse reports* of the widgets s/he has developed. Ratings are statements of liking/disliking a widget (thumbs up/thumbs down), and are described in more detail in Section 3.2.2. Developers are not rated directly. (NB: if abuse reports are severe enough, both the widget and the developer can be "frozen" and/or removed from the system.)
  - A widget does not have a reputation, but it has a *score* based on its a) popularity and b) well-behavedness based on statements made by users, and the reputation of its developer. The widget score may affect the amount of trust user feels towards the widget and affect the trust decision as part of the reputation of the developer.
- Risk is the possibility of a negative consequence that has an impact on a widget user's assets or the achievement of their objectives. The risk involved in installing and using a widget is measured by the impact and probability of negative consequences. Risk must be estimated based on the available information, and it may differ from the actual risk involved. The risk estimate is built on

a) what protected interfaces the widget requires access to and how it accesses them (e.g. what it does with the network connection — as far as this information is provided by the manifest), and b) statements made by users on the widget's behaviour (abuse reports in particular).

- The risk tolerance of the widget user determines how high risks they are willing to take to use widget. It balances the perceived risk, measuring how much incentive the user has to accept it based on his/her risk

attitude[1] and how interesting the widget is to the user. The final decision is not automated, but information may be presented differently based on the user partially expressing his risk tolerance policy through preferences; e.g. a scale of how important it is for the user that his device should not be compromised by a widget, and protect his privacy.

3.1.2 Assumptions and requirements

There are some assumptions made about the operating environment, and some requirements limiting possible solutions.

- Users are not willing to provide their phone number or other clearly identifying information as a basic requirement for registering; this also means that the cost of creating new users is relatively low, and it becomes difficult to limit the number of user accounts a single user can create. Other incentives to hold on to a single account are therefore valuable.
  - Developers are slightly more dedicated to the system and could be asked to provide more information than regular users, e.g., a phone number.
- Widgets will be accompanied by a manifest, which describes what sensitive APIs they will access (and possibly how they will access them, to a degree). This manifest will be used to create an access policy to let the widget access these APIs by the widget runtime environment, which will then stop the widget from accessing sensitive APIs (in ways) which have not been named in the manifest.
- A set of users spend enough time and effort using the widget sharing system that they create a social incentive for themselves to help it thrive by doing volunteer work. This is the basic assumption needed for any system that depends on unpaid labour (minor rewards such as reputation points or some policy-setting power within the system are not objective payment; their value depends on the connection the user has to the system).
- An average developer develops 1–10 widgets during their career.

[1]Risk attitude is partially a general trait of the user and partially dependent of their current situation. Risk-seeking users may e.g. be mostly looking for toys for leisure use on a dedicated device, while risk-averse users may be looking for tools, while being worried about important personal information stored on the device they plan to run widgets on.

- The widgets run on a platform which is regularly connected to the widget sharing system; this allows warnings to be sent to the user about using widgets which have had confirmed abuse reports filed on them.

Requirements

- There is no central trusted party that can be assumed to perform any duties related to the system; the system runs on a platform which is upkept by a party who will not interfere with its workings. (The system should possibly also work if distributed between multiple servers in different administrative domains.)
- The basic browsing, decision-making, downloading, installing and rating activities should be possible to do by mobile phone (as well as optionally with a desktop web browser); this sets some limitations for the complexity of the user interface.

3.1.3 Incentives

An ideal trust / reputation mechanism would include / exclude the following incentives. Incentives for a developer:

- Incentive to develop a widget
  - No incentive to not develop a widget
- Incentive to develop a well-behaving widget
  - No incentive to develop a misbehaving widget
- Incentive to develop an entertaining / useful widget
- Incentive to cooperate with other developers
- Incentive to enter co-developers honestly
  - No incentive to add a fake co-developer (a person who has not participated in the development process)
  - No incentive to leave a co-developer out (not to enter a person who has participated in the development process)
- Incentive to respect co-developers' will (e.g., discuss with them before submitting modifications)?

  - Can we assume that co-developers settle their issues in real life?
- No incentive to remove a useful/entertaining and well-behaving widget
- No incentive to remove and resubmit a widget
- No incentive to abandon a user (developer) account and start anew
- Discouraging creating several/fake user (developer) accounts

Incentives for a user:

- Incentive to use a widget
  - Incentive to make informed decisions on using a widget, taking risk into account
- Incentive to use the widget sharing system
- Incentive to provide an honest rating
  - Incentive to provide a rating
  - No incentive to provide a dishonest rating
- Incentive to send an honest abuse report
  - No incentive to not to send an abuse report if a widget is misbehaving / if the user suspects that the widget might be misbehaving
  - No incentive to send an abuse report if the user does not suspect that the widget is misbehaving
  - Incentive to actively examine/observe a widget for possible misbehaviour?
- Incentive to provide personal information
  - Incentive to provide personal information to gain access to the widget sharing system
  - (Incentive to provide more than the necessary personal information on the widget sharing system, e.g. profile details)
  - (Incentive to store possible personal decision policies in the widget sharing system)
- No incentive to abandon a user account and start anew

- Discouraging creating several/fake user accounts

Other incentives:

- Incentive to handle an abuse report properly (whoever does the handling)
  - Incentive to handle an abuse report in a timely manner
  - Incentive to handle an abuse report diligently (as thoroughly as needed)
  - Incentive to handle an abuse report honestly
- Incentive to carry on moderator activities (e.g., delete inappropriate comments)?
- Incentive to socialize
  - Incentive to provide support to others (e.g., answer questions on the forum / help widget developers solve their problems)

3.1.4 Data sources

The Use cases in Section 2.1 (see also Experimental use cases in Section 2.2) affect what kind of data is available to the reputation / trust mechanisms.

Information about a widget:

- The manifest, i.e. information about what kinds of protected interfaces the widget needs access to.
- Number of downloads (number of registered downloads, possibly also number of users).
- Ratings: thumbs up / thumbs down and a comment
  - Possible detailed ratings: tags (e.g. “entertaining”, “useful”).
- Abuse reports
  - Statistics on the number of widgets using a protected interface, and the number of verified cases of misusing the interface (see Section 3.4).
- Possible trustworthiness statements.
- Possible reviews.

- Possible review ratings.

Information about a user:

- Possible trustworthiness statements.
- The user's risk tolerance policy setting. Users will be able to set their preferred risk level in similar fashion as e.g. the Internet Explorer web browser with a slider moving between e.g. medium-risk–high risk.

3.2 Reputation mechanisms

This section presents a list of general principles to choose from when designing a reputation mechanism, and proposes a mechanism currently being focused on (referred to as *the proposed mechanism*), complemented with some alternatives (referred to as the *alternative mechanisms*).

3.2.1 Principles of reputation mechanisms

The choices general principles of a reputation mechanism depend on the application area and desired/allowed complexity of the system. For widget sharing, the system should be relatively simple in the sense of understandability, as users looking for widgets will generally not be willing to spend considerable amounts of time learning how to use the system.

To clarify the choices made, we note for each position what the corresponding choice has been in the popular consumer-to-consumer auction site, eBay.

The reputation mechanism needs to make decisions on the following aspects (among other things):

- One-user-one-vote principle
  - Opinions of all users are treated equally; *or*
  - some opinions have more weight than others (+ policy for weighing).

In the proposed mechanism and in eBay, the opinions of all users are treated equally in the thumbs up/down ratings. However, moderator-level users have additional power through managing abuse reports, which influence developer reputation and the availability of widgets.

- Zero tolerance for widgets

  - Abuse reports are handled separately from the trust/reputation mechanism and misbehaving widgets are removed from the system (+ policy for removing); *or*
  - misbehaving widgets continue to exist in the system and the trust/ reputation mechanism must model their bad reputation / lack of trust (+ policy for modelling).

Some mildly misbehaving widgets (e.g. buggy, but not malicious ones) continue to exist in the system; when confirming an abuse report, the moderator separately chooses what the consequence of the transgression is, and the mildest consequences do not involve the removal of the widget outright. In eBay, this principle is not really applicable, since the reputation system is reactive only after a sales transaction has taken place. A single transaction has no reputation while it is open. Illegal items for sale can be removed manually.

- Zero tolerance for developers
  - Misbehaving developers (i.e., developers who have developed a misbehaving widget) are removed from the system (+ policy for removing); *or*
  - misbehaving developers continue to exist in the system and the trust / reputation mechanism must model their bad reputation / lack of trust (+ policy for modelling) and they have a change to increase their reputation by behaving well.

Some mildly misbehaving developers continue to exist in the system; see above. The reputation mechanism drops the reputation of a mildly misbehaving developer quickly, but not irrevocably. In eBay, misbehaving sellers/buyers only experience a drop in their reputation. Violating eBay policies can result in more severe measures and affects a special PowerSeller status — although a gold PowerSeller can transgress more than a bronze PowerSeller before this happens [eBa10].

- Lower limit for developer reputation
  - The reputation of a developer cannot be worse than the reputation of a developer who has just registered; *or*
  - the reputation of a developer can sometimes be worse than the reputation of a developer who has just registered (— affects incentive for starting anew with a fresh account).

In the proposed mechanism, the developer's reputation measure can drop below the starting level. This causes some incentive to create multiple accounts. In eBay, it is theoretically possible to get a negative feedback score as well, by receiving more negative ratings than positive (this is rather unusual, however, due to a considerable bias towards positive ratings). In both cases, the reputation information available to the user consists of more than the single score, and for example knowing that the number of ratings given has been 500 positive and 500 negative gives more information about a developer than 0 positive and 0 negative would, even though their reputation measure is the same.

- Separate reputation for widgets
  - A widget can have a reputation on it's own; *or*
  - only developers can have a reputation (all widgets developed by the same developers are then considered equal in this sense).

In eBay, only users have a reputation. In the proposed mechanism as well, only developers can have a reputation. However, widgets have ratings — a set of users' statements of liking/disliking the widget, expressed by counters.

If widgets can have a reputation of their own, the following choice is relevant:

- Lower limit for widget reputation
  - The reputation of a widget cannot be worse than the reputation of a widget (of the same developers?) which has just been submitted; *or*
  - the reputation of a widget can sometimes be worse than the reputation of a widget which has just been submitted (— affects incentive for removing and resubmitting widgets).

The rating counters of a widget begin at 0, so it is possible for a widget that receives only negative ratings to appear worse than a freshly submitted one.

- Transparency of the reputation measure:
  - The reputation mechanism is fully transparent: Users know what measures affect the reputation measures and how (e.g., how rating a widget affects its developers' reputation); *or*
  - the reputation measure is not fully transparent (+ policy for what is hidden).

Note: this considers also incentives. Do the users know (or can they deduce) what incentives they have / what incentives they do not have?

The proposed reputation measure is not fully transparent to the users; overall information of what influences a reputation score is revealed, but the involved constants are not. In eBay, the measure is very simple (the reputation score = positive ratings − negative ratings) and fully transparent.

- Subjective (local) / objective (global) reputation measures
  - The reputation measure depends on the person who is viewing the information (subjective (local)) (+ policy on how this affects the information shown); *or*
  - reputation measures are the same independent of who is viewing the information (objective (global)).

The reputation measure itself is the same independent of who is viewing the information. In eBay, the measure is independent of the viewer.

- Dividing reputation
  - If a widget has multiple developers, the reputation gained by each of them (on average?) is smaller than the reputation gained if the widget had only one developer (e.g., divide the reputation of the widget among all developers); *or*
  - If a widget has multiple developers, the reputation gained by each of them (on average?) is *not* smaller than the reputation gained if the widget had only one developer (e.g., give the reputation of the widget to all developers) (— creates an incentive to declare a friend as a co-developer).

In the proposed mechanism, the reputation gained by each developer per rating is smaller for widgets with many developers since it is divided. This weakens the incentive to add developers to a widget without them having actually participated in the development. The opposite incentive of leaving valid developers out is more easily solved out-of-band amongst the developers.

- Good reputation from one widget
  - Developing only one widget can result in a good reputation for the developers; *or*
  - developing multiple widgets is necessary for a good reputation.

The dynamism of gaining reputation from widgets has two points of interest: when a developer gains ratings from multiple widgets, they can be rated multiple times by a single user, while ratings for a single widget are from unique users. This effect can be controlled by keeping track of ratings from unique users. On the other hand, some widgets are vastly more popular than others, e.g. due to interfacing a highly popular service such as Wikipedia [Wik10], and can skew the reputation scale. Having to a good reputation from multiple widgets strengthens the incentive to produce more than one widget.

In eBay, selling only one item produces a limited amount of ratings which cannot thus contribute to the total reputation very much. On the other hand, one widget can be rated by large number of users. The proposed mechanism can be adjusted so that reputation gained from one widget is limited.

- Range for reputation: what is the range of the reputation measure?
  - $(0, 1)$,
  - $(0, \infty($,
  - $)-\infty, \infty($,
  - other possibilities.

In eBay, reputation can be arbitrarily low or high. In the proposed reputation mechanism, reputation is limited to the interval $(0, 1)$. We believe that the limitation makes the value more meaningful to users; it is more difficult to determine the meaning of "2051" than of "0.7", if the user can assume that the system is configured in a way to divide the reputations of developers on the range relatively evenly.

- Limiting reputation to the range: if the reputation range is not unlimited, how is it made sure that reputation stays within the limits?
  - cut-off (values below the lower limit are changed to lower limit, values above upper limit are changed to upper limit);
  - sigmoid-like functions;
  - other functions where the reputation increases more and more slowly near the upper limit and decreases more and more slowly near the lower limit (the limits are never reached);
  - ranking (e.g., order users by reputation from worst to best, then the worst user gets value 0, the next gets value $\frac{1}{\text{no. of users}}$, the next $\frac{2}{\text{no. of users}}$ etc.); or

– normalizing the reputations of all users so that the sum of reputations is a constant.

The proposed mechanism cuts off values below 0 or above 1. Once the developer's reputation would increase above 1 (or decrease below 0), further increases (decreases) are simply not made. The first event to move the reputation in the opposite direction decreases (increase) the value.

3.2.2 Mechanisms

Proposed reputation mechanism

This section describes the proposed reputation mechanism.

Information used Information about the widget:

- Ratings: thumbs up / thumbs down,
- validated abuse reports.

Developer's reputation Each time a widget receives a thumbs-up rating, its developers' reputation measures are increased by a small constant (α) divided by the number of developers. Similarly, each time a widget receives a thumbs-down rating, its developers' reputation measures are decreased by a small constant (α) divided by the number of developers.

Each time an abuse report about a widget is found to be valid, its developers' reputation measures are decreased by a large constant (β). (This negative effect is the same independent of the number of developers behind the widget; it is not divided by the number of developers.)

The initial reputation of a developer is above zero (we choose 0.5) and it is made sure that the reputation belongs to interval $(0, 1)$ by cutting off other values (if reputation would be above 1, it is set to 1 and if reputation would be below 0, it is set to 0). Having the reputation values be within a fixed range makes the value somewhat more straightforward to present to the user than a value that has no bounds; the user still needs some support to understand if a reputation value of x is for example "rather good" or "average".

The formulas describing this update method are as follows:

- Initial reputation: $\theta_0 = 0.5$. This can be seen as a preconception: without any experience we do not know whether the developer is malicious or honest, so the initial reputation lies between the reputation for malicious developers (0) and the reputation between honest developers (1).

- When a thumbs-up rating is entered: $\theta_t = \max\left(\theta_{t-1} + \frac{\alpha}{\text{no. of developers}}, 1\right)$
- When a thumbs-down rating is entered: $\theta_t = \min\left(\theta_{t-1} - \frac{\alpha}{\text{no. of developers}}, 0\right)$
- When an abuse report is found to be valid: $\theta_t = \min\left(\theta_{t-1} - \beta, 0\right)$,

where $0 < \alpha << \beta < 1$.

Notes and possible modifications:

- All ratings are treated equally (one user, one vote).
- The multiplier α could also depend on how many ratings the rater has given (one user, one vote -principle no longer valid).
- Adding a boring / useless widget can decrease a developer's reputation.
- If more developers are added to a single widget, the total amount of reputation increase per thumbs-up remains the same due to the division — no "free reputation" is awarded for anyone.
- There could also be multiple choices for lowering the reputation depending on the seriousness of the validated abuse report (e.g., bugs versus malicious behaviour).
- There could be a limit for how much reputation can be gained from one widget
  - e.g., reputation gained from one widget could be limited to 0.1.
  - This does not solve the problem that it is difficult to distinguish people who have developed a normal amount of widgets and people who have developed a lot of widgets: it only effects development of k widgets, where $k \times$ limit of reputation gained from one widget ≤ 1. If reputation gained from one widget is limited to 0.1, a developer can hit a reputation of 1 with 5 widgets (assuming the starting reputation is 0.5).
  - The threshold can apply either to the total reputation gain from one widget (shared between multiple co-developers: e.g. 2 developers can get 0.05 each with a limit of 0.1), or the total reputation gain of one developer (e.g. 2 developers can both get 0.1, and the widget "gives out" a total of 0.2 points of reputation).
- The amount of evidence or "certainty" behind a reputation score is not documented in this model.

  - The differences between ratings received by different widgets developed by the same developer can be visualized to the user through showing (in a detailed view) reputation of a developer as an average value from all widgets s/he has developed, combined with the variance between the reputation gained from each.
  - The "variance" of all ratings could be documented as well to bring more weight to ratings that differ from the general trend, which is typically positive.

- There could also be a limit of how many ratings by novice raters / novice users are taken into account.
  - A threshold describing when a user is a novice.
  - Ratings given by novices could be cut off (according to a novice rating limit)
    - e.g., if novice rating limit is 10 and novices have given 50 thumbs-up ratings and 30 thumbs-down ratings, the sum would be 20 but the limited sum is 10.
    - e.g., if novice rating limit is 10 and novices have given 30 thumbs-up ratings and 50 thumbs-down ratings, the sum would be -20 but the limited sum is -10.

- This measure can be represented as a summation corresponding roughly to widget's primary reputation measure wrp_{2b} (choose $\alpha_1 = 0$) in Section 3.2.2.

- Ratings are "temporarily" inherited from the widget, while abuse report affects reputation forever.
  - If a widget is removed, reputation gained/lost through its ratings is removed from the developer's reputation. Reputation lost due to abuse reports remains. See Section 3.1.3.
  - Adding a co-developer to a widget reassigns the reputation between the developers, including abuse report effects.
  - Removing a co-developer (or the account being frozen/removed) should also reassign the reputation among the remaining co-developers; the reputation effect for the removed co-developer is the same as if the widget were removed: effect of ratings is removed, effect of abuse reports remains.

Showing developers' reputation The developers' reputation measures are shown on a widget's page. No combined reputation measure is calculated for co-developers; they are shown separately.

Additional mechanisms Additional mechanisms for filtering out false ratings could be added in top of this mechanisms. The additional mechanisms could, e.g., try to distinguish a group giving lots of similar ratings (possibly fake identities created by a single user) and treat their ratings as given by one person.

Alternative reputation mechanisms

This section contains alternative reputation mechanisms.

Trust flow mechanisms Redefining reputation: The reputation of a widget measures its a) popularity and b) well-behavedness based on statements made by users. It is partially inherited from other widgets developed by the same user. Reputation is divided into primary and secondary reputation. The primary reputation of a widget measures its popularity and well-behavedness based on statements made by regular users. The secondary reputation of a widget also considers the reputation of its developers, "inheriting" some reputation from other widgets built by its developer. The reputation of a developer measures the combined (primary) reputation of the widgets s/he has developed. Information about the widget:

- Number of downloads (number of registered downloads, possibly also number of users)
- Ratings: thumbs up / thumbs down
- Abuse reports

The widget's primary reputation (several alternatives) Primary reputation increases when the widget is downloaded and when it receives a thumbs-up rating. Primary reputation decreases when the widget receives a thumbs-down rating and when an abuse report is sent (verified?).

- $\text{wrp}_{1a} = \alpha_1 \times$no. of downloads$+\alpha_2 \times \frac{\text{no. of thumbs up}}{\text{no. of ratings}} - \alpha_3 * I($there are abuse reports$)$
- $\text{wrp}_{2a} = \alpha_1 \times$no. of downloads$+\alpha_2 \times \frac{\text{no. of thumbs up}}{\text{no. of ratings}} - \alpha_3 *$no. of abuse reports
- $\text{wrp}_{1b} = \alpha_1 \times$no. of downloads$+\alpha_2 \times$(no. of thumbs up $-$ no. of thumbs down$) - \alpha_3 * I($there are abuse reports$)$

- $\text{wrp}_{2b} = \alpha_1 \times$no. of downloads$+\alpha_2 \times$(no. of thumbs up $-$ no. of thumbs down)$-$ $\alpha_3 *$ no. of abuse reports,

where I is the indicator function (1 if the condition is true, 0 otherwise) and $\alpha_1 < \alpha_2 << \alpha_3$.

Notes:

- All ratings are treated equally (one user, one vote)
- wrp_{1a} and wrp_{2a}: A boring / useless (but not misbehaving) widget cannot have a smaller reputation than a new widget (if abuse reports are honest).
- wrp_{1b} and wrp_{2b}: A boring / useless (but not misbehaving) widget may can have a smaller reputation than a new widget (even if abuse reports are honest). (Conflict: this creates an incentive to remove and resubmit a widget.)
- The scale is not fixed; the primary reputation can grow as the number of downloads (and thumbs-up ratings in wrp_{1b} and wrp_{2b} increases. A limited scale can be enforced by cutting of values over the upper limit (e.g., 1).

Reputation of a developer consists of primary reputations of widgets developed by the developer. Only positive primary reputations are taken into account and the primary reputation of a widget is divided equally among its developers.

$$\text{dr}_1 = \sum \frac{\text{no. of widgets developed by this developer} \times \text{wrp}_+}{\text{no. of developers}},$$

where wrp_+ is wrp if wrp > 0 and 0 otherwise.

Notes:

- Adding a boring / useless widget does not decrease developers' reputation (because only positive reputations are taken into account).
- If developers are added, the total amount of reputation remains the same — no "free reputation" for anyone.
- We could take only currently unsolved abuse reports into account.
  - Abuse reports determined to be false alarms could affect the reputation in a positive way. Problem: this creates an incentive to create a fake user and post an abuse report of an own widget. (More work for abuse report handlers; difficult to distinguish real abuse reports.)

Widget's secondary reputation is the sum of its primary reputation and the average reputation of its developers.

$$\text{wrs} = \text{wrp} + \beta \times \sum \frac{\text{developer's reputation}}{\text{no. of developers}},$$

where β is a multiplier depending on the amount of risk (if risk is high, β is low and not all reputation is transferred and if risk is low, β is high and all reputation is transferred).

Notes:

- Including a novice developer decreases widget's reputation (Conflict: contradicts with incentive to enter co-developers honestly).
- If risk is high, information about the widget is more important than information about the developers. Should it be vice versa?

Summary

Table 3.1 summarizes the comparison between the proposed mechanism and eBay according to the listed principles.

3.3 Risk management

This section describes risk management in the widget sharing system.

3.3.1 Basic principles

The relevant definitions from the earlier section:

- The risk involved in installing and using a widget is measured by the impact and probability of negative consequences. Risk must be estimated based on the available information, and it may differ from the actual risk involved. The risk estimate is built on a) what protected interfaces the widget requires access to and how it accesses them (e.g. what it does with the network connection – as far as this information is provided by the manifest), and b) statements made by users on the widget's behaviour, i.e. the reputation of the widget (abuse reports in particular).

Principle	eBay	Proposed
One-user-one-vote?	yes	basic form: yes (exceptions: management options available to moderators), modifications possible
Zero tolerance for widgets (goods)?	not relevant	yes (malicious widgets removed, buggy might continue to exist)
Zero tolerance for developers (sellers)?	no	no
Developer (seller) reputation can be "worse than novice"	yes	yes (can be changed by changing the initial reputation)
Reputation for widgets (goods)?	no	no
Transparent mechanism?	yes	not fully transparent; more complicated
Subjective/objective?	objective	objective
Reputation divided?	only one seller per good	yes
Good reputation from one widget	no	possible to adjust
Range for reputation	$)-\infty,\infty($	$(0,1)$
Limiting reputation	not limited	cut-off

Table 3.1: eBay vs. the proposed system.

- The risk tolerance of the widget user balances the risk involved. It measures how much incentive the user has to accept risks, based on his/her risk attitude[2] and how interesting the widget is to the user. The final decision is not automated, but information may be presented differently based on the user partially expressing his/her risk tolerance policy through preferences; e.g. a scale of how important it is for the user that his/her device should not be compromised by a widget, and protect his/her privacy.

3.3.2 Implementation

Risk impact is deduced from the widget manifest: the amount of possible damage depends on what protected API or API combinations the widget is allowed access to. These are visualized to the user.

The risk probability is deduced from the reputation of the developer (in Section 3.2.2) (validated abuse reports will already either drop it quickly, set it to zero or lock/"remove" the entire developer, so they do not need to be separately considered here), and statistics on the misuse of that particular protected API. For example, *any* widget is verified to abuse the network interface (through the abuse report handling), a counter for the network interface is incremented, and a x misuses out of y total uses in different widgets is stored for each protected API. The developer reputation and the statistical "abuse popularity" of the API are visualized to the user.

The impact and probability are quantified in order to automatically compare to the user's risk tolerance level. The details of this quantification are not set yet.

The user expresses his/her risk tolerance on a high level through e.g. a "paranoia level" slider (either discrete levels or a continuous range, assuming the semantics are neat), and can be allowed to set specific policies separately. Based on the user's risk tolerance, widgets above a certain risk threshold value will either be clearly marked as quite risky (a notification in the widget view) or "hidden": not shown to the user in listings (in addition, widgets found through direct links are difficult to use and require some explicit "I want to see this widget anyway" clicks to see it, and the widget view shows that the risk of the widget is clearly above the user's tolerance threshold).

Note that even if the user has set a high risk tolerance, the risk information elements are visible in the widget view and can be further visualized to all

[2]Risk attitude is partially a general trait of the user and partially dependent of their current situation. Risk-seeking users may e.g. be mostly looking for toys for leisure use on a dedicated device, while risk-averse users may be looking for tools, while being worried about important personal information stored on the device they plan to run widgets on.

users (e.g. by 5-level "scull scales" opposite to 5-star scales): for example, different APIs or their combinations can be (manually) assigned "danger levels" — access to camera alone is inherently less of a privacy risk than access to the network, and access to both the camera and the network increases the privacy risk further.

3.4 Moderators and abuse reports

This section discusses policies and interfaces for handling abuse reports.

3.4.1 Abuse reports

Abuse reports are statements made by a user that a widget misbehaves; either the developer is malicious, or the widget is buggy, and in the latter case the bug can be exploited by a malicious actor or simply look suspicious to a user. There are different levels of abuse, and an abuse report can be truthful and not indicate to a moderator that the widget should be locked ("removed"). Reporting simple bugs or typos as abuse should be discouraged, however. Abuse reports are given by registered users. No technical incentive system encourages abuse reporting in this model; it would be possible to reward good abuse detection somehow, but it must be done carefully to not encourage users to submit reports needlessly. Multiple reports can be given on a single widget; these should be connected and possible to handle as a group. Reports are managed by a special group of users, moderators. A moderator either verifies an abuse report or marks it as invalid. In the case of verifying the report, the moderator selects which (if any) protected APIs the widget misused its access to (for statistics-keeping purposes), and writes a comment to be seen on the widget's view and chooses a further action (from mild to severe):

- None (just the comment).
- Lock the widget (this option can be combined with any of the options below).
- Reduce the developer's reputation by a fixed amount (see Proposed reputation mechanism in Section 3.2.2).
- Nullify the developer's reputation (it is shown on the developer's profile and on the widget's view) — this is not the same as setting the value to 0 once; it *keeps* the value at 0 with no possibility of recovery through getting more positive ratings alone (the nullification can be lifted by moderators, however).

- Lock the developer's user account, simultaneously locking all widgets he has developed.

The comment and notification of the action is forwarded to all users of the widget through the widget platform (and to all users using the developer's widgets, if the developer's reputation / account are affected).

The action can be cancelled by another moderator[3].

Abuse reports are not shown to normal users. It is particularly important to not show the names of abuse reporters to other users, since fear of reprisal can discourage valid reports. If a report is verified to be valid, a moderator may provide a comment to be shown in the widget's view; the report itself is not shown.

3.4.2 Social moderator system

Moderators are a special case of user, and can

- read and manage abuse reports
- remove (and restore) inappropriate comments
- remove (and restore) widgets
- remove (and restore) users.

The different capabilities may be granted to different "levels" of moderator, but in the simplest approach, all moderators have these abilities. Not many moderators are needed in relation to the user base; for example, IRC-Galleria [Irc10], a community-based service for presenting photographs and discussion, has almost half a million users and only ten moderators working full-time [Ruu07]. (IRC-galleria credit pages cite a 9-person "upkeep" staff, and 12 "assistants to upkeep"; it is unclear whether the assistants do moderation work as well.)

Moderator selection should not be fully automated, as moderator access can be used to hijack the system. On the other hand, we have no available trusted parties to manage granting moderator access (including the administrator(s) of the system). This basically means that either moderators select other moderators, or users select moderators. We opt for the first case, since it is

[3]A more complex process involving multiple moderators may be required to cancel an operation than are to set one if the moderator group is divided by disagreements — we consider this relatively unlikely since abuse reports should be relatively clear-cut, but it is a typical problem for e.g. Wikipedia that there is disagreement on what the basis for removing an article as "irrelevant" is.

more difficult to generate corrupt[4] moderators than it is to generate corrupt users.

We assume as starting point that some moderators exist. Further potential moderators are socially discovered, i.e. from discussion forums. Existing moderators make a decision on granting a user moderator access, and to remove moderator access from a fellow moderator by e.g. voting. It would be also possible that a user can grow into a moderator automatically as evaluated by the system, but this is left out of the scope at this point.

Moderators do not need to be developers; developer status is granted automatically based on submitting widgets, and it does not mean that a user is good at detecting misbehaviour in widgets. Automated developer status also means that submitting either a copy of someone else's widget (without getting caught) or a widget that does nothing gives developer status; these kinds of activities should not be actively supported. Instead, technical skills for a moderator can be tested by providing an example of a misbehaving widget and abuse report, and asking a would-be moderator to evaluate the report. The exercise can be automated as well (and can be cheated in), but it measures moderator ability more directly.

Volunteer moderators have less incentive to act fairly and "according to design" than hired moderators, for whom the threat of losing a job is a strong incentive to follow policy. Further, allowing moderators to select new moderators means that if a critical mass of corrupt moderators is reached, this group can recruit an unlimited number of corrupt moderators and override any valid (policy-adhering) moderation decisions. The size of this critical mass depends on the system of selecting new moderators, be it e.g. a requirement that a given percentage of moderators supports the selection, or a given number, or both.

3.4.3 Alternative proposal: Randomized moderator system

When an abuse report is sent, the persons handling it are chosen at random from a set of suitable persons. The chosen moderators then decide whether the abuse report is valid by voting.

Choosing moderators

- Chosen moderators can be developers / non-developers, but with different probabilities; and

[4]We use the term *corrupt* liberally to mean moderators uninterested to act according to a shared policy.

- users who have already downloaded the widget are chosen more probably.

Moderators then carry on handling the abuse report

- Possible to discuss the abuse report?
- If moderators do not respond, other moderators are chosen.

Open questions

- How to identify users capable of handling abuse reports / willing to handle abuse reports?
- Incentive to handle abuse reports?
  - "Normal" moderator systems: moderators have a recognized status in the system.
- Incentive to handle abuse reports properly?
  - There is incentive to handle abuse reports hastily if handling them gives a better status or makes the system "stop nagging".
- When there is no "moderator community", who decides when an abuse report was handled properly?
  - E.g., how to recognize users who claim that they have handled the abuse report properly but in reality they haven't read the source code of the widget?
  - There is no overall responsible group for improving the moderation system.
- If another abuse report is sent on the same widget, should it be handled by the same moderators or different moderators?
  - Same: minimize the effort since the moderators are already familiar with the widget.
  - Different: it's possible that the previous abuse report was not handled correctly (e.g., it was valid but decided to be a false alarm).

3.5 Validation

This section discusses validation of the proposed reputation mechanism and risk management system. The proposed reputation mechanism and risk management system are compared against incentives (see Section 3.1) and misuse cases (see Section 2.4). See also Chapter 4.

Incentives and misuse cases are strongly connected. The Widget Sharing system cannot prevent all misuse cases but it can be designed to include incentives to behave well (not to misbehave) and not to include incentives to misbehave. Some misuse cases can be affected directly at a trade-off affecting other incentives; for example creating multiple identities can be made much more expensive by binding them more closely to real-world identities or group memberships, for example through phone numbers.

Next we describe how the current proposal of the Widget Sharing system relates to incentives (see Section 3.1).

3.5.1 Incentives for a developer

Incentive to develop a widget

- Social incentive, e.g., similar to open source development.
- If the amount of reputation gained from one widget is limited: developing multiple widgets is necessary in order to gain a good reputation.

No incentive to not develop a widget

- Making registration more difficult for developers, e.g., requiring extra information (e.g., phone number or credit card number) may prevent developers from using the Widget Sharing system (instead of other similar systems).
- Received feedback (e.g., user comments and ratings) cannot be guaranteed to be fair. Unfair feedback may affect the developers negatively.
- Abuse report can also be mishandled (e.g., a false alarm may be categorized as relevant) on purpose or unintentionally.

Incentive to develop a well-behaving widget

- If abuse reports are handled honestly, well-behaving widgets contribute to developers' reputation positively. Widgets whose developers have a good reputation probably get more users.
- See also: *incentive to develop a entertaining / useful widget.*

No incentive to develop a misbehaving widget

- Misbehaving widgets may lead to abuse reports which dramatically affect the developers' reputation. Bad reputation of the developers reduces the amount of users willing to use a widget.
- When misbehaving widgets are detected, they are also locked. Users of a locked widget are notified and they probably decide not to continue using the widget. This reduces the damage caused by the widget.
- If the utility gained from a misbehaving widget is large enough (e.g., phishing a considerable amount of credit card numbers), this incentive might exist.
- The system supports the risk awareness of users thus reducing the possible benefits gained from misbehaving widgets.
- Minimizing this incentive requires that
  - users of the Widget Sharing system post abuse reports (depends on skilfulness and devotion of the users)
  - abuse reports are handled quickly and required actions, such as locking the widgets, are taken (depends on the moderator system, see section 3.4)

Incentive to develop a entertaining / useful widget

- The developers gain more reputation in the Widget Sharing system by developing entertaining / useful widgets
- Developers who have developed popular widgets look good in the system

Incentive to cooperate with other developers

- The Widget Sharing system allows entering multiple developers for a widget (and sharing the reputation gained from the widget among the developers)

- If cooperation leads to more entertaining / useful / well-behaving widgets, then developers gain more reputation by cooperating
- Widgets developed with famous developers gain more visibility (appear in the famous developer's profile page)

No incentive to add a fake co-developer

- Co-developers are required to verify their co-developership
- No "free reputation" is given to co-developers: adding a friend as a co-developer reduces own reputation received from the widget.

No incentive to leave a co-developer out

- We assume that this incentive is handled by social rules among the developers.

Incentive to respect co-developers' will

- We assume that this incentive is handled by social rules among the developers.

No incentive to remove a entertaining / useful / well-behaving widget

- Removing a widget removes also good reputation received from it.
- Removing boring / useless / misbehaving widgets is not discouraged.

No incentive to remove and resubmit a widget

- Removing a widget removes also good reputation received from it
- Information about downloads is set to zero and the widget looks less popular
- Malicious widgets:
  - New abuse reports may lower the reputation of the developers again

No incentive to abandon a user account and start anew

- If the reputation of a developer is below the initial reputation, this incentive exists.
- If we require developers to provide more information (e.g., a valid phone number or a credit card number), it makes starting anew more difficult
- Todo: Baton-passing attack

Discouraging creating several/fake user accounts

- No incentive to create several "developer identities" (co-developering with each other) since reputation is divided among developers.
- Fake user accounts used for boosting developer reputation in a simple way (e.g., each fake user rating only widgets of a certain developer, all ratings thumbs-up) could be detected (makes boosting reputation more difficult; more intelligent methods needed)
- Reputation gained by ratings of novice users / novice raters could be small (makes boosting reputation more difficult; each fake user has to provide multiple ratings)

3.5.2 Incentives for a user

Incentive to use a widget

- Downloading and using widgets should be easy. (E.g., widgets can be downloaded without registering in the system.)
- Widget Sharing system should help the user to find useful / entertaining widgets
  - Measuring popularity of widgets
  - Search possibilities, categorization

Incentive to make informed decisions on using a widget, taking risk into account

- Protecting oneself

- The Widget Sharing system should provide information on an easily accessible format

Incentive to use the Widget Sharing system

- Downloading and using widgets should be easy. (E.g., widgets can be downloaded without registering in the system.)
- Widget Sharing system should help the user to find useful / entertaining widgets
  - Measuring popularity of widgets
  - Search possibilities, categorization
- The reputation mechanism and the risk mechanism of the Widget Sharing system should help the user to protect themselves from malicious widgets.

Incentive to provide a rating / comment

- The system should encourage and remind users to give ratings (giving ratings should be easy).
- Giving ratings does not benefit the user directly (in the current proposed system).
- On large scale, giving ratings benefits everyone.

Incentive to provide an honest rating / comment

- If ratings are shown on the users profile page, useful comments make the user look more credible.

No incentive to provide an dishonest rating / comment

- If ratings are shown on the users profile page, stupid comments make the user look less credible.
- One rating changes the reputation of a developers only by a small amount.

No incentive to not to send an abuse report if a widget is misbehaving / if the user suspects that the widget might be misbehaving

- Abuse reports are not shown in public and thus users are protected from revenge.
- Moderators will not punish the user even if the abuse report is not found valid (e.g., a false alarm) unless it clearly is spam / malicious.

No incentive to send an abuse report if the user does not suspect that the widget is misbehaving

- A moderator can lock an account used for sending spam / malicious abuse reports.
- Abuse reports which are not validated are not shown and do not affect developers' reputation (cannot be used for blackmailing).

Incentive to actively examine/observe a widget for possible misbehaviour

- Protecting oneself against misbehaving widgets.
- Social incentive to protect the Widget Sharing community.
- Participating discussions about misbehaving widgets may help gain social status.

No incentive to abandon a user account and start anew

- The user has no reputation measure but the user may have a social status in the system, e.g., if the user has participated discussions in the discussion forum or given comments / ratings.
- If the user account is locked by a moderator (e.g., because the user has sent spam abuse reports), this incentive exists.
- If the user has misbehaved (e.g., by entering stupid comments), this incentive exists.

Discouraging creating several/fake user accounts

- Affecting reputation of a developer (either positively or negatively), see *Discouraging creating several/fake user accounts* from Section 3.5.1.

3.5.3 Other incentives

In this section we examine the social moderator system (see section 3.4).
Incentive to handle an abuse report properly

- Protecting / benefiting the Widget Sharing community
- Gaining social status in the moderator society
- Possible loss of moderator status if an abuse report is not handed properly

Incentive to carry on moderator activities (e.g., delete inappropriate comments)

- Benefiting the Widget Sharing community
- Gaining social status in the moderator society
- Possible loss of moderator status if an abuse report is not handed properly

Incentive to provide support to others (e.g., answer questions on the forum / help widget developers solve their problems)

- Benefiting the Widget Sharing community
- Gaining social status in the Widget Sharing community

Chapter 4

Evaluating the Widget Sharing System

This deliverable discusses possibilities for evaluating and validating different aspects of the Widget Sharing System. The plan for actual evaluations will fall under a future project, and is therefore not covered here.

4.1 Goals of the Widget Sharing System

The research goals for the Widget Sharing System have been presented in the project plan. The desired direction for the introduced Nokia widget platform is to facilitate the creation of a wide selection of innovative widgets, and towards this goal, an open and unbureaucratic development market is considered beneficial. The WiSh project would focus its research on "finding and evaluating a suitable trust model for realising a reputation management system for sharing widgets provided by an open community. The qualitative objective of the reputation system is to encourage users to install and use new widgets by providing a sense of trust".

The trust model requirements are underlined by a change in the security model for widgets. Widgets are installed and run on mobile phones, and they should be allowed access to the operating system and application resources, such as contacts, GPS and the ability to send and receive messages, as well as resources on the Web, through one or several identified hosts or any host. These requirements for widget functionality make the traditional Web browser security model, which relies on sandboxing web programs away from any critical resources, too narrow.

Trusted widgets could be signed by a centralized party, as has been done in e.g. the Symbian operating system. This process has been found too

restricting; the Widget Sharing System should be able to operate without any centralized trusted third party participating in the operation of the system.

These requirements can be further refined based on the high-level design decisions made in the project:

Goal 1 The system should encourage

- the development of innovative and non-malicious widgets, and
- the use of widgets that are interesting (useful, entertaining etc.) and non-malicious.

Goal 2 The system should help the user to make informed decisions on whether to accept the risk the installation and running a particular widget entails. It should not make the final decision for the user, but guide him/her in the process.

Goal 3 The system should collect and disseminate the statements of the users of the widget sharing system on widgets, both on whether they are interesting (useful, entertaining, etc.) and whether they are safe to use or malicious. This information should support the widget user's decision-making and encourage the development of innovative and non-malicious widgets.

Goal 4 The system should be easy and intuitive to use for the target group of users, which is assumed to be the target group for widget use as well: technically inclined owners of advanced (i.e. widget-enabled) mobile phones worldwide.

Goal 5 The system should be built on general principles that are applicable beyond the context of a single Widget Sharing System, i.e. the design decisions should be transparent enough that decisions dependent on the specific nature of the Widget Sharing System can be traced and modified to apply the approach in a slightly different setting.

Goal 6 The costs of deploying the system should be balanced by the benefits gained from it.

4.2 Evaluation methods

The previous section presented the goals for the Widget Sharing System. Each goal and methods for evaluating its success will be discussed in more detail below.

4.2.1 Goal 1: Encouraging widget development and use

The system should encourage its users to develop innovative, high-quality widgets which are interesting to other users. The development of malicious widgets which are detrimental to the users' security or privacy, or attack external actors should be strongly discouraged. In addition, the system should encourage users to use the available widgets to their benefit by making it easy to find interesting widgets and to evaluate the risk involved in installing and using them.

The fulfilment of this goal can be evaluated on two tracks: On one hand, the designers have to build an internal model of how the users think and behave in order to make design decisions, and the decisions and trade-offs should be compared against this internal model. On the other hand, the system targets real users and there are typically some differences between the designers' internal model and the behaviour and thinking of real users. To make these differences visible, *user experiments* are needed.

Comparison to incentives For the first track, we discuss how the desired incentives (see Section 3.1.3) and undesired misuse cases (see Section 2.4) are supported by the system in Chapter 3 (Section 3.5). Since most of the misuse cases of interest cannot be fully protected against in this kind of an open system, supporting the right incentives becomes central.

User experiments For the second track, *interviews/questionnaires* and experiments with *prototypes* that appear operational to the user can be used to find out the user's motivation. The effect the Widget Sharing System design has on the user's motivation specifically can be studied by comparing the results to users' attitudes when shown a different system, e.g. one with no reputation and risk management.

As is typical to user experiments, answers to direct questions are not fully reliable; they should be supported by backup questions measuring the consistency of the answers and possible observation of actual behaviour. Users do not necessarily answer direct questions about their motivations honestly, because they partially involve unconscious processes the user is not aware of (see e.g. McKnight and Chervany's discussion on trust [MC01]), and for social reasons such as wanting to make a good impression by answering questions in a way they believe is "correct", i.e. desired by the person asking the questions.

For gaining a *cross-cultural* point of view, users participating in the experiments can be chosen from different countries, and the results compared.

An opportunity for experimentation with Australian users is opened with Petteri Nurmi's stay in Australia until February.

4.2.2 Goal 2: Supporting the user's decision-making

The system should help the widget user to protect him/herself from bugs and malicious features in widgets by making the user aware of the risk involved in installing and running a particular widget, and supporting an informed decision to accept or not accept them. The system should also respect the user's autonomy, and not force his/her decisions.

This goal has two targets for experimentation. On one hand, the underlying decision support system relies on a model to process data. On the other hand, assuming that the behaviour of the data processing is understood correctly, the suitability of the decision support system for its target users can be evaluated.

Simulations The processing model and its implementation can be evaluated by simulation: giving the system a string of input and observing the output and system state changes. The inputs include a configuration of the system, such as a choice of constants for the reputation mechanism and a choice of quantification of risk impacts. The choice of inputs to experiment on is guided by models of normal behaviour (to observe e.g. how quickly and effectively the system reacts to positive ratings), models of misbehaviour (to observe e.g. how quickly and effectively the system reacts to abuse reports and negative ratings) and models of attacks against the processing system itself (to observe e.g. how resistant the system is to misuse cases such as unfair ratings (Section 2.4.2, page 59)).

User experiments The suitability of the decision support system for its target users can, as noted for Goal 1 (Section 4.2.1), be evaluated initially against supporting the desired incentives (see Section 3.5), and further with user *interviews and experiments*. Example measurements could involve questions considering what information the user takes into account when making the decision, and observation of actual user behaviour — what kinds of decisions the user makes and how they agree with the decision policy the user states s/he is applying. These answers could again be compared to systems providing different supporting information (or none).

It should be noted that the user's decisions are influenced by their trust in the Widget Sharing System besides their trust in single widgets. Trust in the system is built through the general appearance, usability and perceived

"professionality" of the system, and is culturally dependent (see e.g. [FSD+02] on trust in websites and [KCK01] on trust in the Internet).

4.2.3 Goal 3: Collecting and disseminating user statements on widgets

As there is no trusted third party to evaluate and certify widgets, the system depends on the evaluations of the users. This information must be extracted and disseminated to users to both help the decision-making of Goal 2 (Section 4.2.2) and to support the desired incentives expressed in Goal 1 (Section 4.2.1). Towards this purpose, the system should encourage user feedback, collect it and represent it to the widget users in an intuitive and useful way; the feedback should reflect the usefulness and probable trustworthiness of the widgets. Since the underlying web technology makes it possible to create widgets that can modify their behaviour without traceable developer intervention[1], it is crucial that the trustworthiness evaluation is not limited to the current behaviour of a single widget.

The fulfilment of this goal is tightly bound to goals 1 (Section 4.2.1) and 2 (Section 4.2.2) and their evaluation methods. It is also important to simulate the behaviour of the reputation mechanism alone, separate from the risk system, since the processing behaviour of the reputation mechanism is a central part of the mechanism. For the risk mechanism, the modelling and quantification of known risks influences its behaviour strongly, adding complexity into the simulation. The simulation results can be compared to the response of different reputation algorithms available.

4.2.4 Goal 4: High usability

The system should be easy and intuitive to use for the target group of probable widget users: technically inclined owners of widget-enabled mobile phones worldwide. The project has noted the poor usability of various example case sites, and finds good visualization of the available information (for widget discovery and decision-making) crucial to the usability of the Widget Sharing System.

User experiments: User experiments are a central tool for measuring usability. For evaluating the *usability of particular visualization elements*, the elements can be shown to users separately, before they are a part of a

[1] See misuse case Developer-M002: Modifying a widget to be malicious (Section 2.4.2, page 67).

functional-looking full prototype. The users are asked to evaluate how they interpret different visualizations, and how intuitive they find the visualization. For the *overall usability measuring*, a functional *prototype*, or a "paper" prototype where the views are controlled and their behaviour narrated by the evaluator, is used to collect both explicit user feedback and implicit feedback on situations where the user tries to either achieve a goal the system does not support, or tries to use an unsupported approach to achieving an otherwise supported goal.

4.2.5 Goal 5: General applicability

The design decisions of the system follow general principles founded in existing research and related systems in operation. The high-level system should not be tightly bound to this particular instance of widget sharing, and those parts of the system that are dependent on specific attributes of the underlying platform should be identified and made visible.

Comparison to existing systems: The evaluation of the design decisions is supported by discussing which design decisions have been observed to work in existing systems — possibly in a different context — or apply existing research, and which decisions are new to this system.

Further, analysing how the design would change if applied to a different system points out the design choices which are specific to this system. An example target for comparison could be a content aggregation site.

4.2.6 Goal 6: Balancing cost and benefit

The costs of deploying the Widget Sharing System should be in balance by the benefits of the system. Costs include new threats created by the system, configuration and upkeep requirements and added computational and storage costs, while benefits include fulfilment of the goals of the system, threats mitigated by the system and user satisfaction.

Prototypes and testing: The evaluation of the fulfilment of this goal builds on the evaluation of the other goals. In addition, [threat analysis] and [misuse cases] discuss both misuse by the widgets (threats which are mitigated by decision support) and misbehaviour of users enabled in the system (new threats created by the Widget Sharing System). Computational and storage costs can be analysed theoretically by asymptotic complexity, and empirically by load measurements. User satisfaction is difficult to measure before the

system is deployed, but *user experiments on prototypes* combined with *alpha (and beta) testing* can give an indication of the user acceptance and trust attracted by the system.

4.3 Summary

Validation and evaluation methods are strongly bound to the target of validation. Although multiple targets can be evaluated in one experimental setting, the evaluator needs to be aware of all targets to design the experiment accordingly.

This chapter has expressed six goals for the Widget Sharing System, and discussed methods to evaluate their fulfilment. The evaluation approaches can be coarsely divided into three types: analysis (e.g. incentives, comparison, threats), simulation (reputation and risk mechanism behaviour) and user experiments (usability, visualization, incentives). Most analysis methods can be applied before working code of the system exists. Simulation requires the algorithmic portion of the system to exist, while user experiments typically depend on access to the user interface.

Chapter 5

Conclusions

This report has presented research conducted in the Widget Sharing (WiSh) project in its first phase.

An analysis of the use cases, incentives and threats present in the system demonstrated the need for mechanisms to

1. encourage the development and use of high-quality widgets, and
2. to help users reason about the security impact and risk involved in installing and using a widget, and to discourage the development of malicious and insecure widgets.

While these goals are connected, the mechanisms to support them are somewhat different: The first goal is best supported through a form of reputation-based recommendation system, where developers gain social capital through developing quality widgets and users are encouraged to try out and install new software, and give feedback about it. The second goal, on the other hand, demands more control structures, expert analysis of widgets, social costs for developers of insecure or malicious widgets, and in essence discouraging the user from installing widgets that do not seem trustworthy. After presenting structures for both, we have analysed how they influence the incentives of the users and developers.

The evaluation plans define different approaches to validating the approaches presented. The application of these approaches in later phases of the project is outside of the scope of this report, but the division between the two goals has been quite visible in preliminary results already: In user studies, we have found that while recommendations in different forms are generally well accepted by users, there is less motivation for analysing the security and privacy implications of installing a widget. This is particularly challenging for novice users, who may additionally have little experience in evaluating the credibility of information they find online.

The trust of more experienced users is not as easily gained, and they do appear to pay more attention to security information if it is available. Nevertheless, if security is presented in a form of "do not do this" or the nefarious "are you sure?" prompts, it is easily perceived as just getting in the way of users. It is therefore important to introduce security awareness in a positive tone — e.g. recommending more secure widgets is more fruitful than stopping the user from installing their current choice. Designing security in a way that is also usable is an important issue, spanning beyond the application area of widget sharing.

Bibliography

[Amo94] Edward G. Amoroso, *Fundamentals of computer security technology*, Prentice-Hall International, Inc., 1994.

[eBa10] eBay, *ebay powerseller criteria*, `http://pages.ebay.com/services/buyandsell/powerseller/criteria.html`, September 2010.

[FSD+02] B.J. Fogg, Cathy Soohoo, David Danielson, Leslie Marable, Julianne Stanford, and Ellen R. Tauber, *How do people evaluate a web site's credibility?*, Tech. report, Stanford Persuasive Technology Lab, October 2002.

[Irc10] *Irc-galleria*, `http://irc-galleria.net/`, September 2010.

[Jav05] Java Community Process (JCP), *The recommended security policy for GSM/UMTS compliant devices. Addendum to the mobile information device profile version 2.0.1*, October 2005, Available at `http://jcp.org/aboutJava/communityprocess/maintenance/jsr118/MIDP_2.0.1_MR_addendum.pdf`.

[KCK01] Kristiina Karvonen, Lucas Cardholm, and Stefan Karlsson, *Designing trust for a universal audience: A multicultural study on the formation of trust in the internet in the nordic countries*, Proceedings of the First International Conference on Universal Access in HCI (UAHCI'2001) (New Orleans, LA, USA), August 2001.

[KKI09] Kristiina Karvonen, Theofanis Kilinkaridis, and Olli Immonen, *Widsets: A usability study of widget sharing*, Human-Computer Interaction – INTERACT 2009 (Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Prates, and Marco Winckler, eds.), Lecture Notes in Computer Science, vol. 5727, Springer Berlin / Heidelberg, 2009, pp. 461–464.

[LdBSV04] M. S. Lund, F. den Braber, K. Stølen, and F. Vraalsen, *An uml profile for the identification and analysis of security risks during structured brainstorming.*, Tech. report, SINTEF ICT, May 2004.

[MC01] D. Harrison McKnight and Norman L. Chervany, *Trust and distrust definitions: One bite at a time*, Trust in Cyber-societies: Integrating the human and artificial perspectives, vol. LNCS 2246/2001, Springer-Verlag, 2001, pp. 27–54.

[RK05] Sini Ruohomaa and Lea Kutvonen, *Trust management survey*, Proceedings of the iTrust 3rd International Conference on Trust Management, 23–26, May, 2005, Rocquencourt, France, Lecture Notes in Computer Science, vol. 3477, Springer-Verlag, May 2005, pp. 77–92.

[Ruu07] Tiina Ruulio, *Poliisi saa vihjeitä moderaattoreilta*, ITviikko (2007), no. 15.11.2007.

[She10] Yiyun Shen, *The simulation of reputation systems in widget sharing communities: A comparison study*, Master's thesis, University of Helsinki, Department of Computer Science, January 2010, Available at `http://cs.helsinki.fi/u/yshen/Thesis/MasterThesis_yshen.pdf`.

[SVB06] Andreas Schlosser, Marco Voss, and Lars Brückner, *On the simulation of global reputation systems*, Journal of Artificial Societies and Social Simulation **9** (2006), no. 1.

[Sym05] Symbian Software Ltd., *Symbian OS v9 security architecture*, February 2005, Available at `http://library.forum.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-35228542-8C95-4849-A73F-2B4F082F0C44/sdk/doc_source/guide/platsecsdk/SGL.SM0007.013_Rev2.0_Symbian_OS_Security_Architecture.doc.html`.

[Wik10] *Wikipedia, the free encyclopedia*, `http://www.wikipedia.org/`, 2010.

Appendix A

Visualization

This section describes the various visualization needs in creating an understandable, easy-to-use user interface for passing the underlying information to the user. Both mobile user interface and web user interface issues will be dealt, as these have different capabilities and needs.

There is a need to think of visualisation both for desktop web browser and for mobile device. The limited space of the mobile device is especially challenging. This means, among other things that not all features, e.g. writing reviews will probably be meaningful to be provided in the mobile user interface.

A major lack in current visualisations of current reputation systems is the poor understandability of scale behind the ratings and reputations, and what they consist of, i.e. their ingredients. Providing scale and easy access to more information within minimal amount of clicks are key in correcting these flaws. However, on the main user interface level, especially on the mobile device, a simple, immediately understandable visualisation is also needed in order to allow for quick but accurate decision-making on whether the widget (and its developer) can be considered as trustworthy.

Another factor for making the system easy to use is to provide exemplary cases of how information provided is meaningful, i.e. what is its significance to the user. These examples can be in literary form and/or graphical presentations on the consequences of the selections and decisions made at each step.

The visualisation needs include at least the following:

Widgets

- score (ratings: preferably show number of positive and negative ratings instead of positive − negative etc.),

- affected interfaces and their combinations (in downloading phase, what about later on?),
- update available,
- review exists,
- discussion forum new entry (optional),
- abuse report(s) exist,
- abuse report(s) verified,
- widget stained, and
- widget frozen.

Risk

- no risk
- risk scale, e.g. 1–5 of 5 skulls to indicate danger

User Ingredients:

- user behaviour,
- user interaction,
- user generated-content,
- site reputation, and
- user reputation.

What's needed for user visualisation:

- Different tools for different users: visualise different types of users in a different way: user status user/developer/moderator.
- Engage the most enthusiastic users directly: give moderator status; "user of the month" -status.
- To enhance volunteering: Give users incentives and recognition on basis of activity: reputation, e.g. 1–3 of 3 stars.

Information about users: user profile (desktop only?)

- user activities/history list (desktop only?)

Moderator

- Moderators should be able to communicate: moderator discussion forum, moderator comments visualised differently on general discussion forums. Possibility to search for moderators from user database.
- Level of notification: abuse report under review, abuse report rejected, abuse report verified.
- Editing versus deleting.

Appendix B

Widget capabilities

Below, we analyse relevant widget capabilities from the point of view of Symbian user capabilities [Sym05] and the Mobile Information Device Profile (MIDP) Specification for using Java on embedded devices [Jav05].

The set of relevant capabilities is not fixed anywhere yet. It depends on e.g.:

- technical things, e.g. mapping to underlying OS like Symbian capabilities; and
- which kind of device access i.e. JavaScript APIs make sense (e.g. no smart card access will probably ever be there).

Following are of interest as a reference: Symbian and MIDP. Symbian capabilities are quite coarse but maybe they are a good set as a starting point. MIDP "function groups" are actually designed to be displayed/prompted to user. There are functions that are not so relevant here.

Most realistic would probably be:

- network (HTTP) access,
- write user data (address book, calendar, pictures, ...),
- read user data,
- location, and
- user environment i.e. multimedia recording.

B.1 Symbian "User capabilities"

B.1.1 Mapping real-world permissions — User Capabilities

The process of identified these capabilities was a lengthy and involved one. First we attempted to identify those accesses that require policing and then tried to map those requirements into something that is meaningful for a user. In addition, more capabilities means greater complexity and complexity is widely recognised as being the chief enemy of security. A solution based on capabilities should therefore seek to minimise the overall number deployed. These map fairly broadly onto the main threats which are unauthorised access to the outside world and preserving the confidentiality/integrity of user data. We assert that any permission requirement can be expressed in terms of some combination of those. In a sense, these capabilities represent orthogonal groupings because they police different kinds of access together. The capabilities identified today are as follows:

NetworkServices Grants access to remote services without any restriction on its physical location. Typically, this location is unknown to the phone user. Also such services may incur cost for the phone user. This typically implies routed protocols. Voice calls, SMS, internet services are good examples of such network services. They are supported by GSM, CDMA and all IP transport protocols including Bluetooth profiles over IP.

LocalServices Grants access to remote services in the close vicinity of the phone. The location of the remote service is well-known to the phone user. In most cases, such services will not incur cost for the phone user. This typically implies a non-routed protocol. An application with this capability can normally send or receive information through Serial port, USB, IR and point-to-point Bluetooth profiles. Examples of services are IR beaming with the user's PC, Bluetooth gaming, file transfer.

ReadUserData Grants read access to data belonging to the phone user. This capability supports the confidentiality of user data. Today, Contacts, messages and calendar data are always seen as user confidential data. For other content types such as images or sounds, it may depend on context, and ultimately be up to the application owning the data to define.

WriteUserData Grants write access to user data. This capability supports the management of the integrity of user data. Note that this capability

is not necessarily symmetric with ReadUserData. For instance, one may wish to prevent rogue applications from deleting music tracks but may not wish to restrict read access to them.

Location Grants access to the live location of the device. This capability supports the management of user's privacy regarding the phone location. Location information protected by this capability can include map co-ordinates and street address, but not regional or national scale information.

UserEnvironment Grants access to live confidential information about the user and his/her immediate environment.This capability also protects the user's privacy. Examples are audio, picture and video recording, biometrics (such as blood pressure) recording.

B.2 MIDP

A device with a small display may not be able to present all permissions on an API level to the user in a single configuration settings menu in a user friendly manner. Therefore the device is not required to present all individual permissions for user confirmation. Rather, a certain higher-level action triggered by the protected function should be brought to the user for acceptance. The high level functions presented to the user essentially capture and reflect the actions and consequences of the underlying individual permissions. These so-called function groups are as follows:

Network/cost-related groups

Phone Call the group represents permissions to any function that results in a voice call.

Call Control the group represents permissions to any function that results call set-up or tear-down of a *restricted network connection*(?).

Net Access the group represents permissions to any function that results in an active network data connection (for example GSM, GPRS, UMTS, etc.); such functions must be mapped to this group.

Low Level Net Access the group represents permissions to any function that results in an active low level network data connection (for example Sockets, etc.); such functions must be mapped to this group.

Messaging the group represents permissions to any function that allows sending or receiving messages (for example, SMS, MMS, etc.)

Restricted Messaging the group represents permissions to any function that allows sending or receiving messages to a restricted messaging service (for example, Cell Broadcast, etc.)

Application Auto Invocation the group represents permissions to any function that allows a MIDlet suite to be invoked automatically (for example, push, timed MIDlets, etc.)

Local Connectivity the group represents permissions to any function that activates a local port for further connection (for example, COMM port, IrDA, Bluetooth, etc.)

Authentication the group represents permissions to any function that gives a MIDlet suite access to authentication functionality.

User-privacy-related groups

Multimedia recording the group represents permissions to any function that gives a MIDlet suite the ability to do any kind of multimedia recording (for example capture still images, or to record video or audio clips).

Read User Data Access the group represents permissions to any function that gives a MIDlet suite the ability to read a user's phone book, or any other data in a file or directory.

Write User Data Access the group represents permissions to any function that gives a MIDlet suite the ability to add or modify a user's phone book, or any other data in a file or directory.

Smart Card Communication the group represents permissions to any function that gives a MIDlet suite the ability to communicate with the smart card.

Location the group represents permissions to any function that gives a MIDlet suite access to Location information.

Landmark the group represents permissions to any function that gives a MIDlet suite access to Landmark information.

Whenever new features are added to MIDP they should be assigned to the appropriate function group. In addition, APIs that are specified elsewhere (that is, in other JSRs) but rely on the MIDP security framework should also be assigned to an appropriate function group. If none of the function groups defined in this section is able to capture the new feature and reflect it to the user adequately a new function group *must* be defined in this document by requesting an update to this document from MSA or MIDP as appropriate.

If a new function group is to be added, the following should be taken into consideration: the group to be added *must* not introduce any redundancy to the existing groups, the new group *must* be capable of protecting a wide range of similar features. The latter requirement is to prevent introducing narrowly scoped groups. The new function group *should* be sufficiently future-proof to contain new features added by future APIs and should not only concern the features being initially included in it.

It is the function groups and not the individual permissions that should be presented when the user is prompted. Furthermore, it is the function groups that should be presented to the user in the settings of a given MIDlet suite.

This report studies the environment and requirements for a platform for sharing small applications for mobile phones among users. This widget sharing platform has an important role in supporting users in determining which widgets are trustworthy enough to install and use: The applications are not certified by a central trusted party, and they have access to various resources on the phone, such as the address book, camera, phone calls and an Internet connection.

Helsinki Institute for Information Technology HIIT
Tietotekniikan tutkimuslaitos HIIT (in Finnish)
Forskningsinstitutet för Informationsteknologi HIIT (in Swedish)

The Helsinki Institute for Information Technology HIIT is a joint research institution of Aalto University and the University of Helsinki for basic and applied research on information technology.

Its research ranges from fundamental methods and technologies to novel applications and their impact on people and society. HIIT's key competences are in Internet architecture and technologies, mobile and human-centric computing, user-created media, analysis of large sets of data and probabilistic modeling of complex phenomena.

HIIT works in a multidisciplinary way, with scientists from computer, natural, behavioural and social sciences, as well as from humanities and design. The projects are conducted in collaboration with universities, companies and research institutions.

http://www.hiit.fi

HIIT Technical Reports 2010-3

ISBN 978-952-60-3543-7 (electronic)
ISSN 1458-9478 (electronic)