<table>
<thead>
<tr>
<th>Document type</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>D9.3.3 — Social Software Security: Developer Guidelines</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP9</td>
</tr>
<tr>
<td>Deliverable Number</td>
<td>D9.3.3</td>
</tr>
<tr>
<td>Editor(s)</td>
<td>Dave Clarke, KU Leuven</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>Public</td>
</tr>
<tr>
<td>Preparation date</td>
<td>4 July 2013</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Legal Notice**

All information included in this document is subject to change without notice. The Members of the IWT SBO SPION project make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the IWT SBO SPION project shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.
The IWT SBO SPION Project

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Participant name</th>
<th>Country</th>
<th>Department</th>
<th>Participant role</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K.U. Leuven</td>
<td>BE</td>
<td>COSIC/ESAT</td>
<td>Coordinator</td>
</tr>
<tr>
<td>2</td>
<td>K.U. Leuven</td>
<td>BE</td>
<td>DISTRINET</td>
<td>Partner</td>
</tr>
<tr>
<td>3</td>
<td>K.U. Leuven</td>
<td>BE</td>
<td>DTAI</td>
<td>Partner</td>
</tr>
<tr>
<td>4</td>
<td>K.U. Leuven</td>
<td>BE</td>
<td>ICRI</td>
<td>Partner</td>
</tr>
<tr>
<td>5</td>
<td>Vrije Universiteit Brussel</td>
<td>BE</td>
<td>SMIT</td>
<td>Partner</td>
</tr>
<tr>
<td>6</td>
<td>University of Ghent</td>
<td>BE</td>
<td>OWK</td>
<td>Partner</td>
</tr>
<tr>
<td>7</td>
<td>Carnegie Melon Univer-</td>
<td>USA</td>
<td>Heinz</td>
<td>Partner</td>
</tr>
</tbody>
</table>

Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Dave Clarke</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>2 Rula Sayaf</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>3 Willem De Groef</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>4 Frank Piessens</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>3 Brendon Van Alsenoy</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>4 Ralf De Wolf</td>
<td>Vrije Universiteit Brussel</td>
</tr>
<tr>
<td>5 Jo Pierson</td>
<td>Vrije Universiteit Brussel</td>
</tr>
</tbody>
</table>

External Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Steven Van Acker</td>
<td>KU Leuven</td>
</tr>
<tr>
<td>2 Philippe De Ryck</td>
<td>KU Leuven</td>
</tr>
</tbody>
</table>
Table of Contents

1 EXECUTIVE SUMMARY ................................................................. 5

2 DEVELOPER GUIDELINES............................................................. 6
   2.1 GENERAL GUIDELINES .......................................................... 6
   2.2 PROGRAMMING GUIDELINES .................................................. 7
   2.3 APPLICATION ARCHITECTURE ................................................. 8
      2.3.1 Web Application ............................................................. 8
      2.3.2 Backend API ................................................................. 9
      2.3.3 Third Party Plug-ins ..................................................... 10
   2.4 CORE SECURITY CONCERNS ............................................... 11
      2.4.1 Authentication ............................................................. 11
      2.4.2 Access Control ............................................................. 12
      2.4.3 Contextual Privacy ...................................................... 13
   2.5 LOGGING AND DATA MANAGEMENT POLICY .......................... 14
   2.6 INTERACTION WITH USERS ............................................... 15
      2.6.1 Privacy Settings ........................................................... 15
      2.6.2 I/O ................................................................................. 15
      2.6.3 User Posts ................................................................. 15
      2.6.4 User Accounts and Passwords ....................................... 16
   2.7 MISCELLANEOUS ............................................................... 16

Keyword List

social software, security, developer guidelines
1 Executive Summary

According to the Description of Work the objectives of Deliverable 9.3 is to evaluate the prototypes. Deliverable 9.3.3 forms part of this larger deliverable. The objective of this is to provide:

- Guidelines and recommendations towards developers and SNS providers.

In the document we present such a list of guidelines for developers of social software, including both new social software platforms, but also plug-ins to existing platforms. The guidelines are based on research done thus far during the SPION project, and were elicited by considering a number of different perspectives—the legal perspective, the social perspective, the user perspective, and the developer perspective—although the guidelines themselves are presented thematically. The themes the guidelines are organised around are as follows:

**general** — guidelines all programmers should consider

**programmer** — fundamental programming-related guidelines that should be observed to improve the quality of software

**application architecture** — guidelines related to the architecture of a social software system, covering topics web application, backend API, and third-party plug-ins

**core security concerns** — guidelines specifically concerning security of the social software, namely, authentication, access control, contextual privacy

**logging and data management** — guidelines concerning the storage of data for purposes such as accountability and auditing and based on European legal regulations.

**interaction with users** — guidelines for dealing with users, including topics such as privacy settings, I/O, user posts, user accounts and passwords; and

**miscellaneous** — additional guidelines on the interaction with search engines and facial recognition software.

**Readers Guide:** The deliverable consists of a single part, namely a collection of guidelines along with additional explanations aimed at developers of social software sites, including online social networking sites, and developers of plug-ins or apps for leveraging existing sites.
2 Developer Guidelines

Social software allows users to connect, interact and share personal data (thoughts, images, etc.) either through a web-based application or indirectly through one of the myriad of mobile devices currently available. Such users entrust the social software provider with their personal data and thereby enter an implicit and often ill-understood agreement with the provider that the data is properly handled. It is well documented, however, that users’ personal data has been abused, and such abuses have the potential to negatively impact the social software provider’s reputation and ultimately its business. Thus it is of interest not only to users but also to the social software provider to ensure that their application is implemented securely and preserves user privacy requirements.

The SPION project has been investigating techniques for improving security and privacy of the users of social software, from a multidisciplinary (educational, technical, legal and social) perspective. To help developers avoid such violations of privacy, this document provides a number of guidelines for developers of social software, whether of stand-alone systems or plug-ins to existing social software platforms. By following these guidelines, developers will be able to construct social software sites (and plug-ins to existing sites) that respect the (often implicit) privacy requirements of users, and to make users aware of how their data is used. In addition, developers will be made aware of common implementation mistakes that open the door to potential attacks on their infrastructure. These guidelines are an outcome of the SPION Project. This project will continue for another two years, resulting in, among other things, and updated version of these guidelines.

The guidelines are organised topically, as follows: general guidelines (Section 2.1), programming (Section 2.2), application architecture (Section 2.3), core security concerns (Section 2.4), data management and logging (Section 2.5), user interaction (Section 2.6), and miscellaneous (Section 2.7).

Certain guidelines are marked with a ‡ to indicate that although they aim to improve user security, they are susceptible to abuse which could result in privacy violations.

2.1 General Guidelines

These guidelines are of a general nature; they should be observed by any social software development effort.

1. Keep in mind that security should always be a concern.

If an attack is not made on your application directly, it may use your application to leverage another attack. Such an attack may involve data collected about the users of the social software. The business model of social software entails that the social software provider benefits most from protecting user data.

2. Keep in mind that privacy should always be a concern.

Social software data handles personal data. The data is highly sensitive and users trust the software provider of not violating their privacy as stated in the privacy policy agreement (see Point 5).

The very nature of online software vulnerabilities and attacks on web infrastructure changes rapidly over time, so developers need to be up-to-date with common vulnerabilities and exploitable software errors:

3. Monitor OWASP’s list of the top 10 web application vulnerabilities1 and CWE/SANS’s top 25 most dangerous software errors.2

---

1https://www.owasp.org/index.php/Top_10_2013-Top_10
2http://www.sans.org/top25-software-errors/
Pay attention to common programming mistakes by carefully monitoring specific lists provided by organisations and companies dealing with software security. These lists typically contain frequently made mistakes and vulnerabilities for distinct types of applications (e.g., web applications). These lists can help to raise security awareness within developer teams. As an extra, these lists often contain good advice on how to prevent and fix all kinds of issues.

4. **Make your (the developer's) perception of the privacy problem and solutions to it explicit, in particular consider how they are phrased in terms of users’ experience.**

Defining a privacy problem is often the starting point for developing solutions. Defining a problem, however, is a very normative process. Formulated differently, technology is steered by the perceptions of the developer, but such a techno-centric definition of the privacy problem is often removed from the user experience of the problem. Transparency is required: does the developer define privacy as a fundamental right or as a commodity to be traded? Is the developer aware of why users perceive things as privacy issues?

5. **Provide an explicit privacy policy.**

The policy should make explicit how data will be handled and what the ramifications are. It should be considered as a contract between the social software provider and its users.

### 2.2 Programming Guidelines

The next set of guidelines govern general programming concerns. These are relatively easy to observe; they work towards patching the most common vulnerabilities.

6. **Be as defensive as possible: code will fail.**

Code is always executed within a specific environment. Even the simplest assumptions about this environment can and will fail when in production, for a variety of reasons: e.g., hard disk failures, network congestions, and overload of requests. Making code robust to any type of failure will at least make sure that the application fails in a controlled way.

7. **Compiler warnings are there for a reason: take them into account.**

Compiler warnings are typically about non-functional issues: mixing up signed and unsigned comparisons, declaring unused variables. In general, they warn you about behaviour that is different than generally expected. Code that compiles without warnings, isn’t necessarily safe code. Code that doesn’t, is almost surely not safe.

8. **Never neglect a return value.**

Return values of functions and methods are frequently used to return error codes (e.g., in C/C++) to its caller. By carefully checking return values, one can ensure that failure is safely handled.

9. **Avoid buffer overflows.**

Use automatic bounds checking to avoid out-of-bounds behaviour, as such errors are the cause of a vast majority of software vulnerabilities.

10. **Avoid integer overflow or wraparound.**

Integers express a bounded set values, and thus can overflow (or wraparound) when computations exceed those bounds. Computations that exceed those bounds can lead to strange behaviour. You need to pay specific attention to the use of integers to avoid such overflows, especially when they are triggered by third-party code.

11. **Don’t reinvent the wheel: write as little code as possible and reuse existing libraries as much as possible.**
By using existing libraries and software as much as possible, you can limit the amount of implementation errors. Typically, these software packages are used (and thus tested and maybe even reviewed) by far more users and developers than your own custom written one. Apart from the security benefits, these libraries are often written by people with in-depth knowledge of the specific functionality, resulting in optimised and robust code.

12. **Never copy-paste code found on the Internet.**

People often put pieces of code on web sites (e.g., to explain specific features or to make a case). Most authors don’t have the intention to provide the readers with a full-blown piece of code, but merely with its bare intention. This boilerplate code can be used, but should be modified and enhanced to incorporate most other programming guidelines specified within this document. In general, be careful about including third-party scripts in your web pages. Preferably, evaluate the security of the script and then download it and place a copy on your own web server.

13. **Do not try to implement your own cryptographic algorithms.**

Developing high-quality, provably-secure cryptographic algorithms requires a large degree of expertise. Stay away from custom crypto code.

14. **Isolate third-party content as much as possible (e.g. iframe, sandboxing)**

Since third-party content is potentially untrusted, it should be isolated in a separate context. This prevents the untrusted code from interfering with the web application, or from abusing the application’s permissions.

15. **Prevent unwanted framing (e.g. x-framing options).**

A framing attack enables a malicious application to trick the user into clicking on certain actions in the target application (i.e. click-jacking). X-Framing-Options prevents unwanted framing, and thus reduces the risk for potential framing attacks.

16. **Separate presentation logic from code.**

Separating presentation logic (HTML) from code (JS) and markup (CSS) is a good housekeeping practice. This reduces the risk of mistakes, and enables the use of security policies such as CSP.

17. **Consider using frameworks that include build-in security checks.**

There are many examples. Some from the research world include WebDSL, mobl, ur-web; consider also commercial frameworks.

### 2.3 Application Architecture

These guidelines concern the overall architecture of the social software system, including the web application itself (Section 2.3.1), the back end API available to third-party applications and plug-ins that access the social software, for instance, via a web service (Section 2.3.2), and guidelines regarding the development of third-party plug-ins (Section 2.3.3).

#### 2.3.1 Web Application

If you value your users and their privacy, protect it. This fundamentally requires protecting the web application underlying the social software site from common vulnerabilities.

18. **Use HTTPS whenever user credentials (e.g., passwords, but also session cookies) or other**

---

3Hypertext Transfer Protocol Secure.
privacy-sensitive data are transmitted, and deploy site over TLS\(^4\) using HTTP strict transport security (HSTS)\(^5\) when your website uses SSL/TLS so that visitors’ browsers know to always contact the website over an encrypted channel.

Any attacker who is able to observe traffic between a user and a website can steal user credentials (username/password, user credentials, ...) when they are transmitted in plaintext (i.e. without HTTPS). Sites dealing with user credentials and sensitive info should be deployed over SSL/TLS. Unfortunately, this still allows SSL stripping attacks, which can be eliminated with HSTS. HSTS forces the browser to load a site over SSL/TLS, eliminating SSL stripping attacks.

19. **Use fresh, random CSRF tokens to avoid CSRF\(^6\).**

A CSRF attack allows an attacker to trick the user’s browser to make requests in the user’s name. Developers should add hidden random token fields to forms to prevent CSRF attacks. By adding a secret token to a form, a website can ensure that the request initiated from the intended source, typically a form within the web application. Ensure that the tokens are randomly generated each time and ensure that the infrastructure is available to make sure that tokens are valid, both for the application and for each specific session.

20. **Deploy anti-XSS\(^7\) measures such as CSP\(^8\).**

Cross-site scripting vulnerabilities give an attacker complete control over the web application in the user’s browser. These vulnerabilities should be avoided at development time. As a second line of defense, CSP prevents inline scripts (i.e., injections) and limits the sources from where scripts are loaded, effectively minimizing the remaining attack vectors in case of an XSS vulnerability.

21. **Ensure that their cookies are tied down: use Http-Only and Secure cookies.\(^9\)**

Cookies are typically used for session management, but are easily stolen through Javascript or a network attack. By using Http-Only and Secure over HTTPS, an attacker is prevented from stealing the cookies, thus can no longer take over the session of an authenticated user.

22. **Disallow URL redirection to untrusted or unknown sites (‘Open Redirect’).**

An open redirect takes a parameter and redirects a user to the parameter value without any validation. This is used in phishing attacks to get users to visit malicious sites without realizing it.

### 2.3.2 Backend API

The following guidelines are relevant whenever your social software platform provides an API for third party applications and plug-ins. Such an interface could be via a web service.

23. **Avoid defining permission on a set of database tables. Instead, define the permission on a table level.**

For example, a permission to access friend_profile should only give access to the basic profile data of friends and not to the data in the tables: friend_profile, friend_list, friend_album.

24. **Define permissions on the column level of database if possible, and employ views to restrict the dataset available to third party applications.**

\(^4\)Transport-layer security.
\(^5\)http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security
\(^6\)Cross-site request forgery.
\(^7\)Cross-site scripting
\(^8\)Content Security Policy http://www.w3.org/TR/CSP/
This further restricts the data accessible through the API and relies on database access control mechanisms to enforce protection, rather than custom access control code, which may not be as robust as one hopes.

25. **Define permissions in a restrictive way to avoid unintended data leakage.**

For example, when one column is a foreign key in another table and an application has access to latter table, the foreign key leaks data about the first table. The application should be granted access to the table except the foreign key column that is shared with a table the application has no permission to access.

26. **Ensure that third party applications cannot access the data of users who have not themselves given permission for the application provider to do so.**

It is the social software provider’s responsibility to specify mechanisms to enforce asking for the required minimal permissions only.

27. **Design Application Programming Interfaces (APIs) to support granular access control, so that access to user data can be limited to that which is necessary for an application to function.**

The most privacy friendly applications and plug-ins use only the data required to function and no more. Provide support for your third-party developers to limit their data requirements.

### 2.3.3 Third Party Plug-ins

The following guidelines are relevant when developing a third party plug-in or app for an existing social software platform, such as Facebook, Google+, and so forth.

28. **Select the minimal permissions required to run your application.**

Users tend to trust the social software and may extend this trust to external applications. Consequently, they may not question the permissions applications require without knowing the relationship between the permissions and the functionality.

29. **The application should still function even when a subset of its requested permissions are not granted.**

The application should be written in such a way that the denial of a particular permission should not affect all of its functionality. Default values, such as a default user profile, can be used in place of actual user data.

30. **Incentivise applications to request a minimal set of permissions.**

The ease of accessing data may attract applications to ask for permissions more than their functionality requires. The business model of the social software provider may depend on controlling who can access the data, and thus the provider should encourage the applications to ask only for the required permissions.

31. **Define fine-grained permissions for third party applications.**

Users do not know the providers of third-party applications. It is therefore the responsibility of the social software provider to verify the purpose of data access by such applications. Giving users fine-grained control over what applications can access decreases the chances that their privacy is violated.

32. **Require third-party applications to justify the permissions they asks from users, and how they relate to its functionality.**

Users should have sufficient information in order to make well-informed decisions about the permissions applications ask for—this guideline will help satisfy that user requirement.
33. Mediate between permission requests by the user and the application.

The involvement of the provider ensures the genuine granting of requests by the user to the application. An application may ask for more permissions than needed if the provider is not monitoring the user-application agreement. The permissions request should be handled through social software backend.

34. Ensure that third party applications, particularly those related to privacy management, properly integrate with the existing social software platform.†

To create a seamless experience for the user thereby increasing the chance that application will be actually used in the way they are meant to be, the application needs to be fully integrated into the host platform. If the tool claims to manage privacy, then privacy-relevant information must be made available to the tool.

2.4 Core Security Concerns

The following set of guidelines address core security concerns regarding user authentication (Section 2.4.1), access control (Section 2.4.2), and the use of contextual information in helping users make privacy-relation decisions (Section 2.4.3).

2.4.1 Authentication

These guidelines concern authentication: how do you know your users are who they claim to be.

35. Properly authenticate users to your application. Allow two-factor authentication where possible.

Two-factor authentication strengthens authentication by requiring a second factor after the username/password stage. Since a password is something that a user knows, ensuring that the user also has some other authentication channel thwarts attackers that steal or gain access to passwords.

36. Do not store keys in a database.

To avoid break-ins if the database is breached, store the keys only in the memory of the web server. Alternatively, split the key and store parts in different locations, such as in a database, in a file and so forth.

37. Use per-user salt when storing passwords.

Salting password hashes ensures that passwords are more difficult to recover in case of a leak or breach.

38. Require users to re-authenticate whenever they want to access sensitive features.

This avoids problems that could occur if a user forgets to log out from a session and leaves their computer accessible to whosoever comes along.

39. Periodically, outdate sessions.

This ensures, among other things, that when a third-party accesses a computer, it will not have active sessions to social software.

40. Monitor where users log in from and detect anomalies, perhaps by informing users and asking them to confirm, perhaps via an offline channel (email, SMS).†

Note that this gives rise to potential privacy concerns regarding user tracking.
2.4.2 Access Control

The following guidelines concern access control: which parts of the data available through the social software are accessible to which users.

41. Do not rely on presentation-layer access control.

   For example, hiding a hyperlink does not prevent a user from going to the link URL — they may inspect the source of the web page.

42. Ensure that users can exercise control over their data in a granular fashion, but refrain from introducing excessive complexity.

   Users need to be able to set up their access control policy in a manner that suits their needs, but users are typically not excessively computer savvy, so complexity needs to be limited. See also Point 68.

43. Enable users to review the history of granted permissions and accesses to their data.

   The ability to review how data was accessed allows the user to assess the appropriateness of access to their data and potential privacy threats. This assessment contributes to enhancing future access control-related decisions.

44. Ensure that users have the ability to revoke or modify granted permissions at any given time.

   The appropriateness of a specific permission may change over time, thus users should have the flexibility to change them whenever they need to.

45. Provide various types of mechanisms to identify and manage the audience of various posts.

   Studies have shown that many users group their friends according to membership to some community related to an institution (e.g., a school) and some structure within that (e.g., a class). Other users, however, use different ways of managing their friends and the audience of their posts.

46. Configure the default privacy settings in a way that:
   - restricts access to profile data to contacts selected by the user
   - requires the user to explicitly authorize any further access to profile content, as well as access to traffic or location data.

47. Use P3P\textsuperscript{10} to define access control policies for both the users and the provider.

   With P3P, the provider can define policies about how data is intended to be used, and users can define policies about how their data can be processed.

48. Link data items to the user who originally posts them via the software, the original subject

   Users mostly post personal data via the software. This data should be linked to the user who posts it to facilitate controlling it according to her own privacy preferences.

49. Verify whether a newly posted item has been posted by another user before linking it to the currently-posting user.

   This can help avoid content stealing and related privacy concerns.

50. Verify whether a newly posted item contains information about other users or relevant subjects, e.g., a photo of a group of people.

   Relevancy to other users can be detected using image and text analysis techniques to identify authors. Such mechanisms can be trained on the data of the existing users.

\textsuperscript{10}http://www.p3p.org
51. **Allow subjects to specify access control policies targeted towards others users, third party applications, the social software provider, and external parties such as other social software.**

Users need to be able to control all potential access to their data, by all other parties involved. This control is subject to what is entitled by law.

52. **Allow the relevant subjects of a data item to participate in specifying the access control policy over this item in collaboration with the original subject.**

When a data item concerns some users other than the one uploading it, the formers should be involved in defining the appropriate access control policy over that item.

53. **Allow users to specify access control policies over relational or behavioural data, such as their relationships within the social softer or their information about their interaction with other users.**

Users generate various types of data using the social software. The access control should apply to such types of data.

54. **Always consult the access control policy of a data item specified by its original subjects.**

A data item can be copied by users and shared without the permission of its subject. The social software should facilitate tools to check for such dissemination, and enforce the policy of the original subject upon detection.

### 2.4.3 Contextual Privacy

Research indicates that users have difficulty in knowing their audience and underestimate their audience size, which could ultimately lead to privacy problems. This can be helped by better managing the context of the posts. The following concerns relate to the incorporation of appropriate context information in order to help users make informed privacy-related decisions.

55. **Make the interaction context explicit by (optionally) making the following visible to the audience of post: the data subjects, data attributes of the post, and its audience.**

When the audience know the subject of a data item, they can better identify the interaction context. In certain situations, anonymity is required to protect the privacy of subjects. A proper solution to balance between the owner's anonymity and an identifiable context is to use pseudonyms or other anonymisation mechanisms that could protect against strangers, but keep them identifiable to their audience.

Automatic text analysis tools can help with extracting and making some data attributes visible. The visibility of potential audience can reveal information about the subject’s preference of audience and disclosure patterns. Should the subject be aware that the potential audience will be visible to others then his selection of the potential audience will be well-informed. If the subject is not aware of the visibility then this causes frustration due to unfamiliarity with the social norms in the social software.

56. **Avoid ambiguous context by making the extended audience visible.**

In Facebook (June, 2013), when any of the audience becomes a subordinate subject, her friends become part of the audience too. The friends of the subordinate subject are called the extended audience. The extended audience should be visible to the original subject and the rest of the audience to avoid context ambiguity.

57. **Avoid ambiguous context by connecting the offline to online context.**

For example, allow a user to add the location and event a photo was taken at.
2.5 Logging and Data Management Policy

Social software needs to collect and store user data in order to deliver its promised functionality. Additional data may also need to be collected in order to fulfil legal requirements or to retroactively audit potentially undesirable activity.

58. **Inform users about their legal rights and responsibilities towards others’ data.**

Most users are ignorant of their rights and responsibilities. Social software sites can contribute to educating users.

59. **Inform users about the parties that can control their data according to the legal system.**

For example, the social software provider is the data controller under the EU Data Protection Directive.

60. **Inform users in a simple-and-easy-to-understand manner about the purposes of collecting and processing their data, unless obliged by law not to.**

Transparency is the best business model to ensure user satisfaction.

61. **Publish an accurate privacy policy describing what your site/app does with collected data.**

For instance, clearly inform user what will be done will do with gained permissions (e.g., geolocation).

62. **Follow the European Directives on Data Retention.**

Compliance to the European Directives will ensure that your business reduces legal problems if violations of user data occurs.

63. **Verify, where possible, whether a newly posted item is legally protected.**

Posting of protected data, such as copyrighted images and videos, should not be permitted.

64. **For reasons of audit and accountability, the application should maintain a secure log of all activities.**

Should any untoward activity occur using the social software, a properly recorded log will enable an accurate, temporal play-back of activities as they occurred, thereby making it easier to appropriately assign blame. This includes recording deleted items precisely, both when they were created and when they were deleted.

65. **Users should be explicitly informed about what data is logged for auditing purposes.**

Again this fits with the general motto of transparency.

66. **When a subject is being audited, the access control policies of this subject will not be enforced if they affect the auditing.**

In a sense, the auditing procedure operates above the regular operation of the social software. This also implies that access to it must be carefully guarded.

67. **Allow users to specify a validity period for the data item they share. Upon expiration, the data item should be inaccessible to any party except for the auditing mechanisms.**

Users should be able to delete their data, including by anticipating ahead of time that they will want to delete their data. Such data should become no longer visible, though it needs to be retained in the secure log.
2.6 Interaction with Users

These guidelines concern interaction with the user, ranging from dealing with privacy settings (Section 2.6.1), user input and output (Section 2.6.2), posts related to users (Section 2.6.3), and user accounts and passwords (Section 2.6.4).

2.6.1 Privacy Settings

These guidelines deal with user privacy settings, which inform the social software about user preferences regarding the dispersal of their data across the social software and beyond.

68. Create a clear, simple and logical privacy settings pane that can easily be manipulated by users.

This is particularly challenging as privacy policy models need to be expressive to cover a sufficiently wide range of possible user requirements, yet the privacy settings pane needs to be simple enough to easily cover most common base cases and intuitive enough to deal with the complex settings.

69. Ensure that no changes can be made to users’ privacy settings without their explicit authorisation.

Changing user privacy settings without their knowledge not only leads to possible violations in their privacy, but creates a negative perception of the social software, which may ultimately hurt the business.

See also Section 2.7 concerning the interaction of the social software and (external) search engines.

2.6.2 I/O

The following guidelines concern data originating from or sent to users.

70. Never trust data originating from the user. Regard all user input as hostile, so do input filtering using whitelists, not blacklists.

An application cannot distinguish between a harmless user and an attacker. Therefore, it makes sense to treat all users as attackers and consider all their inputs to be attacks. Data from the user may consist of embedded SQL or Javascript aimed at disrupting the operation of the social software, including deleting database tables. To avoid this malicious input from interfering with your application, test it for legal values and refuse any odd-looking input. More specifically, use parametrised, prepared SQL statements instead of concatenated ‘dynamic’ queries to avoid SQL injection attacks.

71. Output sanitisation/validation is as important as input sanitisation/validation.

Although input validation is important for any application, output sanitisation is equally important, especially for web applications. Web applications often output a mix of data and (JavaScript) code, which will execute on the client side. By not sanitising output, an attacker may be able to trick the end user’s browser to execute the output as code, allowing the attacker to compromise the client's browser. Use the proper escape syntax for the kind of client-side code they are generating (i.e. the correct escape syntax for HTML, JavaScript, CSS, ...).

2.6.3 User Posts

Users need to have control of the kinds of content that can be posted in their profile and the kind of content that is posted about them.
72. **Enable users to decide which (types of) entities can post content to their profile.**

Users may allow others to contribute to their profile, for instance, by posting on their wall, or they may limit such contributions. In particular, the user should be able to control postings by third party apps. The types of contributions should also be controllable.

73. **Notify subjects of when content reposted by another user. Give the subjects the opportunity to approve the reposting.**

Users have the right to control the dissemination of posts concerning them. The social software should track when the user is the subject of a post and when such posts are shared or reposted. Users should then have control over whether the repost is allowed.

74. **Enable users to pre-moderate comments of other users before being published on their profile.**

In order to avoid slanderous or offensive messages, spam, and other undesirable content, do not allow posting to user accounts without prior moderation. To alleviate the potential, subsequent problems with the user experience, introduce a trust mechanism which will allow users to bypass such moderation. For instance, users can nominate certain of their friends as trusted, or users can earn trust over time via ‘approved of’ interaction. Furthermore, users can flag undesirable activity, resulting in a bad reputation for the posters involved. Such mechanisms fall under the name ‘trust and reputation systems’.¹¹

### 2.6.4 User Accounts and Passwords

This section provides a few guidelines for dealing with user accounts.

75. **Encourage users to pick strong passphrases.**

Studies have shown that weak passwords are easy to guess and strong passwords are hard to remember. Even passwords that seem strong using techniques such as substituting ‘0’ for ‘o’ are not as strong as expected, as password crackers can easily employ these substitutions. Passphrases are stronger and more memorable.¹²

76. **Provide an exit-strategy for users.**

Allow users to leave with all their data. Furthermore, if this website provides long-term user accounts, be prepared to have users that pass away from old age, the ultimate exit.

77. **Protect against fake/sybil user accounts and spam.**

Fake accounts are used by spammers to broadcast their message to as many of your users as possible. Avoid the creation of these fake accounts by using CAPTCHA and regularly monitoring user accounts for odd behaviour.

### 2.7 Miscellaneous

Following are a number of guidelines that did not find a home in the previous sections.

Search engines will traverse your site, collecting information and making it available publicly, thereby potentially violating the privacy measures you intended to put in place. Developers must guard against these.

78. **Ensure that users have the ability to restrict the visibility of (some or all of) their profile against search engines, including those internal to the social software itself.**


The developer should ensure that ‘restricted access’ (i.e., non-public) profiles are not discoverable through search engines unless explicitly authorised by the user concerned, including via direct links that may be posted on external forums and so forth.

79. *Ensure that use of facial recognition software is disabled by default.*

Photos and videos of users can be analysed using facial recognition techniques. The developer should give fine-granular control to users over when on which data such techniques can be applied. The default privacy-friendly setting is to have the recognition disabled.