Peer-based Complex Profile Management

Mark Wallis, Frans Henskens and Michael Hannaford

Abstract The rising popularity of Web 2.0 applications has seen an increase in the volume of user-generated content. Web Applications allow users to define policies that specify how they wish their content to be accessed. In large Web 2.0 applications these policies can become quite complex, with users having to make decisions such as 'who can access my image library?', or 'should my mobile number be made available to 3rd party agencies?'. As the policy size grows, the ability for everyday users to comprehend and manage their policy diminishes. This paper presents a model of policy configuration that harnesses the power of the Internet community by presenting average-sets of policy configuration. These policy "profiles" allow users to select a default set of policy values that line up with the average case, as presented by the application population. Policies can be promoted at an application level or at a group level. An XML approach is presented for representing the policy profiles. The approach allows for easy profile comparison and merging. A storage mechanism is also presented that describes how these policies should be made persistent in a distributed data storage system.

1 Introduction

Security and privacy are key concerns as the Internet continues to evolve. The introduction of Web 2.0 [5] has seen an increase in web applications that rely upon user-generated content. When a user uploads content to a web application s/he is relying on that application to protect the data, and to only use it in ways to which the user has agreed. Social Networking websites often provide a "profile" [7] that can be configured to describe how a user wishes the web application to use their data. As the complexity of these sites increases, the profiles are becoming larger

Distributed Computing Research Group, University of Newcastle, Australia, e-mail: mark.wallis@uon.edu.au

and hence more difficult to configure. Users currently have two options available to them in regards to configuring their profiles:

- Rely on the default values provided by the web application.
- Review every configuration value manually and choose an appropriate setting.

The first option requires the user to rely on the web application provider to select a default value that is in the best interest of the end user. This has been proven to not always be a reliable option [4]. The second option is a labour-intensive task that relies on the end user understanding each presented profile option and selecting an appropriate answer. As the size of the profile configuration grows, there is an increased tendency for users to ”give up” and not complete the full configuration. This leaves many options at their default settings and the user in the same situation as those who take the first option.

There is a clear need for solution that allows quick, easy configuration of profile options, while also allowing users the ability to tweak options to their specific needs. Every user of a web application implementing such configurable user profiles has the same policy creation problem; it therefore makes sense to leverage the user population to provide a solution. This paper presents an approach that relies on surveying user profiles to create an average set of profile responses. This average set is summarised at multiple levels of the user hierarchy, from connected groups to an application-wide view. These profile averages are presented as pre-determined ”policies”, and allow the user to rely on average case configuration data based on what their peers have previously configured.

This paper provides advances in the following areas:

- An XML language definition for representing and storing policy sets.
- A process for averaging and comparing policy sets in a hierarchical manner.
- A storage platform for user policies based on a distributed storage mechanism.
- A mechanism for detecting policy structure change, including the creation of new policy options.
- Methods of enforcing policy configuration options, based upon a centralised storage mechanism, at a company level.

The remainder of this paper begins by presenting a framework for storing policy information. Section 2 reviews previous research in this area. Sections 3 and 4 deal with the way policy is created and distributed between the web application and the end user. Section 5 presents a distributed storage model for policy sets and associated meta-data. Sections 6 and 7 define the way security is enhanced by this solution, and how group management fits into the design. Finally, Sections 8, 9 and 10 present a proof-of-concept implementation and summarise the advances presented in this paper.

2 Previous Work

The benefits that Social Networking can contribute to various security and privacy functions have previously been investigated. For example Besmer, et.al [1] provide a recent survey of the ways input from social networks can affect policy configuration choices made by end users. Their research suggests that, in particular, the visual prompting must be sufficiently strong for any recognisable effect to be effected. The complexity of policy configuration in Social Networking applications has also been previously presented [3], along with reviews on the importance of users' awareness of the affects of privacy policy configuration [6].

This previous research led the authors to recognise the importance of privacy configuration, with an emphasis on the need for a solution that is scalable, easy to use, and secure. Previous work in this area does not provide a complete technical solution, nor does it address issues such as policy storage, or how such policies can assist with overall data security. This paper presents a technical solution to these problems.

3 Profile XML

The first step in automating policy configuration is to define a method for storing "profiles". A profile is a set of policy answers specific to an entity, an entity being a user, or a group of users. The various questions that need answering by a particular policy are specific to each web application. These questions take the form of meta-data related to the profile.

The Web Application would first make an XML document, describing the options that need to be specified by a user's policy, publicly available. An example of such meta-data is given below:

```
<profile_meta>
 <application>
  <name>Social Networking Website</name>
   <address>www.social.com</address>
 </application>
 <policy>
  <privacy>
   <option name="MobilePhoneAccess">
    <answer name="Public" desc="Public Access"/>
    <answer name="Group" desc="Friends Access"/>
    <answer name="Private" desc="Private Access"/>
   </option>
   <option name="EmailAccess">
    <answer name="Public" desc="Public Access"/>
    <answer name="Group" desc="Friends Access"/>
```

```
    <answer name="Private" desc="Private Access"/>
   </option>
  </privacy>
 </policy>
</profile_meta>
```

Entity-specific profiles matching this meta-data are then stored using matching XML, as presented below:

```
<profile>
  <entity>
    <entity_name>Mark Wallis</entity_name>
    <entity_type>User</entity_type>
  </entity>
  <application>
    <name>Social Networking Website</name>
    <address>www.social.com</address>
  </application>
  <policy>
    <privacy>
      <option name="MobilePhoneAccess" value="Public"/>
      <option name="EmailAccess" value="Group"/>
    </privacy>
  </policy>
</profile>
```

The above XML format is scalable to suit multiple policy options grouped in a hierarchical manner. Ensuring that all entities store their profile options in the same way provides benefits when it comes to comparing and averaging profile data. For example, a specific profile can be validated against profile meta-data to ensure that all question/answer elements are provided. This allows applications to upgrade and 'version' their profile meta-data, while retaining backwards compatibility with existing user profiles. By way of demonstration, a web application may choose to add a new privacy configuration option as shown below:

```
<profile_meta>
  <policy>
    <privacy>
      <option name="HomeAddressAccess">
        <answer name="Public" desc="Public Access"/>
        <answer name="Private" desc="Private Access"/>
       </option>
    </privacy>
  </policy>
</profile>
```

A simple SAX parse of the application's presented profile meta-data, could be used to compare it to the user profile. This would show that the profile was missing

a (Profile/Policy/Privacy/Option@name=HomeAddressAccess) element, and hence was out-of-date compared to the current policy information being requested by the application.

While XML has been chosen as the storage mechanism in this specific implementation, the concept is generic and could equally be implemented in other languages, such as JSON.

4 Profile Creation and Operations

Now that a way of storing profile information has been defined we, must provide a way for the user to create their personal profile XML.

To achieve manual creation of profiles, the web application would present a user interface that collects profile information from the user. This information is then made persistant in the XML format described above. Such a manual solution is based on the typical solution we currently see used by Social Networking applications. As previously discussed, manual profile creation is unwieldy, so the solution proposed by this paper allows the user to auto-generate a profile based on profiles averaged from the population base of the application.

Previous work presented by Besmer, et al [1] introduces visual cues that indicate the most common answer to each policy question to the end user. Our XML schema approach allows these common suggestions to be auto-generated at multiple levels. For instance, the user may wish to select defaults based on the entire population, or perhaps based only on the average generated when surveying the ”friends” in their social network. The definition of a typical, or ’majority average’ view is highly user-group specific.

Using an XML language allows for advanced operations such as merging of, and comparison between, profiles. XPath [2] operations can be used to extract singular values from multiple policy sets, average the results, and generate a new policy using the average value. Entities can compare polices, using basic XML comparison techniques, to highlight differences in their configuration. Strict enforcement of XML schema will allow multiple concurrent sequential parsers to efficiently compare numerous policies without a large memory footprint. By way of comparison, current systems that rely solely on user-interfaces for policy definition, do not give end users access to the above style of operations.

A key benefit of this XML approach focuses on the meta-data. Web Applications will, as time goes on, need to change the policy information they collect from their users. For example, a new feature may be added that introduces a new policy option. Such change enacts a change to the profile meta-data XML document, and because of the document structure it is easy to compare the meta-data to a user profile, and to see which new options need to be completed by the user. Again, the user may choose to simply accept a default for any newly added options.

5 Profile Storage

Profile meta-data is generated and stored by web applications themselves. According to the current Web 2.0 design, user profile information is also stored by web applications. This is sub-optimal for a number of reasons, including:

- Users have no protection against web application owners changing policy settings without their (the users') knowledge.
- Comparison, and other profile operations, must be implemented by the web application.
- End users are not able to look at their profile configuration in any way other than that supported and allowed by the web application.

These limitations demonstrate the benefits of offloading the storage of profile information to an entity controlled by the end-user. User-controlled distributed storage models [9] provide a platform for storing the profile information at a data storage provider chosen by the end user.

In a distributed content storage system (DSS) the user manages, or procures, a data storage service that is used to store all their user-generated content. Web Applications 'subscribe' to pieces of information held in these data stores, and offload the responsibility for storing and securing the data to the data owner. Enhanced client software dynamically builds pages out of framework code provided by the web application, combined with data provided by numerous distributed content storage systems. This model solves issues relating to data ownership, data freshness and data duplication. The user can rely on their own data storage service being the single-version-of-the-truth for all their data [9].

Accordingly, web applications such as social networking websites would request read-only access to a user's profile information via a web service interaction with that user's data storage service. Any updates to policy meta-data would be pushed from the web application to the storage service using a publish/subscribe [10] operation, at which time the end-user would be notified that their policy needed review. Figure 1 shows how the distributed storage solution works in relation to the end user, and to the web application.

When profile information is stored in DSS, the fact that the storage is controlled by the end-user removes the risk that web application owners may change policy information without the user's consent. Objects requested from the distributed data storage service may be cached, with updates available using a publish/subscribe mechanism. This allows the web application to request a user's profile upon user login, without having to constantly re-request the full profile each time a policy setting needs to be evaluated.

6 Data Security

The same DSS used to store the user's profile is used to store the user's generated content, such as their image library and contact details. Additional policy security can be implemented at the DSS level. For instance, if a user stores their mobile phone number in the DSS, and their policy permits only private access to this data, the DSS can enforce the policy at either the data linkage or data access stages, so that a badly-implemented (or rogue) web application is not able to act against the user's wishes.

7 Group Management

For profiles to be aggregated at multiple levels, we need a way to store a user hierarchy. The storage location of this hierarchy will greatly influence which component of the overall system is responsible for generating and storing aggregated profile data.

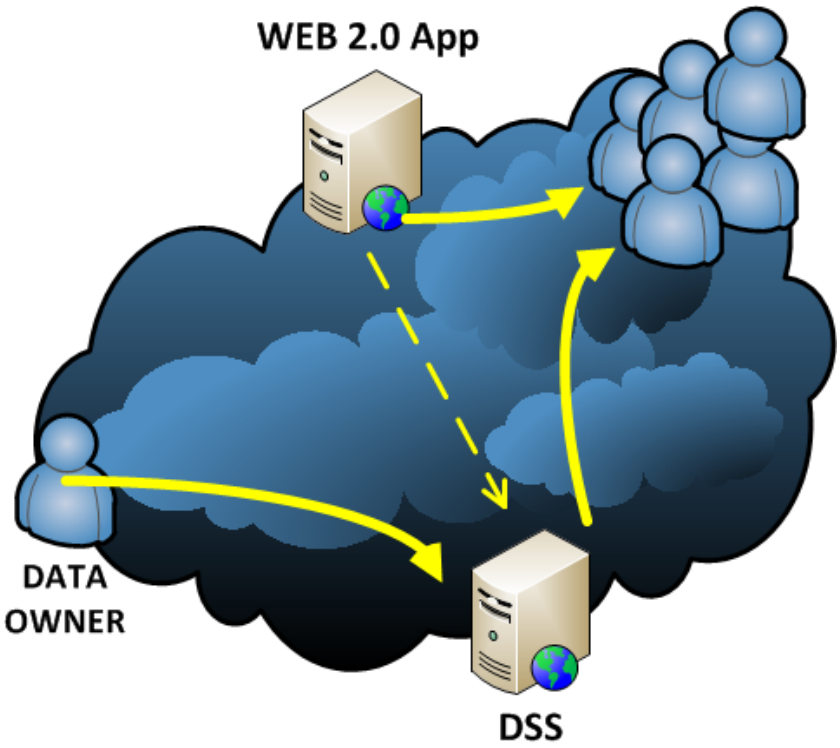

Fig. 1 Distributed Profile Storage Model

The use of a distributed data service arrangement for storing profiles ensures that no one data store (which might otherwise have, as a side effect, performed aggregation) is responsible for provision of all user profile data. Web applications will continue to access in-memory cached versions of profiles, which they could use to maintain running averages across their respective user populations. For example, each time a user logged in, data could be added to the average set stored locally by the web application. The result is that more frequent users of the web application will have greater influence on the average data set, while the impact of users who have not logged into the application for long periods of time would eventually become negligible.

Alternatively, a web application that required aggregation based on the entire (rather than recently active) user base, could regularly request policy information directly from the users' respective DSSs for the purpose of aggregation. It should be noted, however, that this is not scalable for large user populations.

Another possibility is that the DSS could store the user hierarchy data. Continuing with the social networking example, it is clear that the way a particular user relates to others in the user community represents a valuable piece of information. If the user hierarchy were stored in the distributed store, it could become the DSS's responsibility to query the DSSs of the user's friends to generate the average sets. This distributed architecture leads itself to a more component-based approach [8] to web application development.

8 Implementation

A proof-of-concept implementation of the XML profile schema, and associated operations, has been implemented within previously-reported systems relating to distributed content storage [9]. XML schemas were persisted on a user-managed storage service, and accessed via a web service from a prototype address book application. The following sample policy options were defined:

- Mobile Phone Access - private, group or public
- Email Address Access - private, group or public

Operations were created to summarise those values at both group and an application wide levels. Users were then given the option to automatically accept defaults at either level, and then to alter those settings if they wished.

This implementation demonstrated the feasibility of removing responsibility for policy storage from the web application. Caching was implemented within the web application and distributed storage framework, thus ensuring that the policy file was only fetched from the DSS on user login. This reduced any performance impact that would otherwise be introduced through DSS fetches every time a policy-based decision was made by the web application. Policy data storage was also able to take advantage of existing DSS features, such as push notifications. This allowed for policy updates to occur mid-session, if required. Use and observation of the

implementation indicated no adverse performance implications associated with the benefits afforded by the combination of distributed, XML-expressed profiles.

9 Conclusion

This paper presents an XML-based technique for representing user policy information. The structure of this representation allows for efficient operations, such as comparisons between policies, and averaging of policy sets.

By separating user policy storage from Web 2.0 applications, we added a level of user protection against service providers' changing and evolving policy without appropriately notifying their users. The storage of policy information in a user-specified distributed storage server allows policy information to be applied across multiple web applications. This provides the users with the possibility of a 'single version of the truth' for all their policy decisions.

The presented techniques add overall data security to a user's data, while also addressing the issues of scalability and performance. The way user relationships (or 'connections') are stored in Web 2.0 systems, is also discussed.

10 Observations and Future Work

Storage of multiple policies in a user controlled storage service creates the possibility of a unique identifier that identifies a user across multiple services. For example, at present a user would have a unique ID with his/her social networking website, and that would differ from the unique ID used by his/her photo library website. Centralised cross-application profile management suggests that the storage server should generate and propagate a single identifier for each user across the multiple systems accessed by the user.

Privacy concerns with such a model obviously need to be reviewed. In the absence of such a centrally provided unique identifier, we are seeing the ID used by popular social networking websites (e.g. Hotmail or Facebook login) becoming defacto choices for unique identifiers - with smaller websites allowing you to 'log in' using your social networking profile. The effect of this, including the automatic propogation of security profiles, warrants further investigation.

References

[1] Besmer A, Watson J, Lipford HR (2010) The impact of social navigation on privacy policy configuration. In: Proceeding of the Sixth Symposium on Usable Privacy and Security

[2] Clark J, DeRose S (1999) Xml path language (xpath). W3C URL http://www.w3c.org/TR/1999/REC-xpath-19991116/

[3] Lipford HR, Besmer A, Watson J (2008) Understanding privacy settings in facebook with an audience view. In: Proceedings of the 1st Conference on Usability, Psychology and Security

[4] McKeon M (2010) The evolution of privacy on facebook. Personal Website URL http://www.mattmckeon.com/facebook-privacy/

[5] O'Reilly T (2005) What is web 2.0. O'Reilly Net URL http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[6] Strater K, Lipford HR (2008) Strategies and struggles with privacy in an online social networking community. In: Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 1

[7] Vickery G, Wunsch-Vincent S (2007) Participative Web And User-Created Content: Web 2.0 Wikis and Social Networking. Organization for Economic

[8] Wallis M, Henskens FA, Hannaford MR (2010) Component based runtime environment for internet applications. In: IADIS International Conference on Internet Technologies and Society (ITS 2010)

[9] Wallis M, Henskens FA, Hannaford MR (2010) A distributed content storage model for web applications. In: The Second International Conference on Evolving Internet (INTERNET-2010)

[10] Wallis M, Henskens FA, Hannaford MR (2010) Publish/subscribe model for personal data on the internet. In: 6th International Conference on Web Information Systems and Technologies (WEBIST-2010), INSTICC