Databasket: Coherent Shared Data for the Web

Michael Bernstein Max Goldman

December 14, 2007

Introduction

Web applications have put significant effort into personalization services to improve the user experience. The current personalization model suffers from two major drawbacks: each site has access to a very limited subset of information about the user, and the users themselves have little or no control about what data is maintained and how it is kept private. Users thus repeat personalizing rituals across a number of sites, specifying their names, email and shipping addresses, and interests; and web sites often make poor predictions, recommending items when inappropriate or the wrong items altogether. Web sites occasionally see privacy gaffes such as America Online's in 2006, sharing personal data on the Web and exposing their users to fraud and identity theft.

In this paper we propose a user-controlled central database of personal information called Databasket (Figure 1) as a potential reinvention of web personalization. We place the data locally on the user's computer, ensuring that the user him- or herself has primary control over how the data is shared. We provide a Javascript API for web sites to query over a range of this data once the user has granted permission, thus allowing web sites access to customize using broader, more up-to-date data. To control data access, we have designed an interface drawing on research in usable privacy and security to keep the user (arguably the most vulnerable link) aware and in control.

To follow, we introduce the Databasket system and its design. We focus first on related work in centralized personal data repositories for the web. Then we describe a typical Databasket use scenario, the system's user interface and developer API, and back-end implementation. We report on a first-use study of the interface using two web sites developed using the Databasket API, and finally focus on challenges and future work for the system.

Figure 1: Databasket running in Mozilla Firefox. The user is currently granting permission for the Random Friend Generator web site to access his or her friend information.

Related Work

Web personalization has achieved a large amount of commercial and research attention in recent years. Original research in web customization focused mainly on adapting site navigation (*e.g.*, [9]). Commercial web sites like Amazon and Netflix have found success with collaborative filtering techniques [21], building a model of user preferences and then making predictions based on what similar users found useful. Customization efforts may also be initiated by the user, such as with Greasemonkey [4] or Chickenfoot [11], where the user can write site-specific scripts to customize rendered web pages after the fact. Databasket does not focus on end-user customization, but instead attempts to give web developers access to the data they can use to personalize the web.

Databasket builds upon a wide variety of work focusing on open data standards and shared information. The vision of the Semantic Web [10] spells out a web with information freely shared in machine-readable formats, with information decentralized but universally accessible. Recently researchers have proposed radically new architectures for the web to achieve

similar goals [18]. Reality has tended toward web sites maintaining private data sources, however, so *scrapers* such as Sifter [15] have been developed to reverse-engineer data in *deep web* sites such as Amazon. Whether scraped or obtained via web sites with public APIs, this data and its accompanying mashup culture is beginning to see research in tools for scripting and manipulating (*e.g.*, [13]).

Centralizing the user's data in a single point of contact is another driving force behind Databasket. Piggybank [14] is such a browser extension, allowing users to download and maintain a wide variety of web data provided it can be translated into a Semantic Web format. Such ideas have also been explored in the Semantic Web community, termed "semantic caching" [17]. Google Gears [3] and the upcoming SQLite installation in Firefox 3 offer client-side web databases, but segment data by a domain origin security model rather than letting multiple sites view and edit the same data. The OpenID system [6] centralizes a small amount of user data to allow universal login across web sites. Tools such as Haystack [20] have explored the user interface ramifications of having all data interrelatable.

Databasket derives many of its design principles from work in usable security (*e.g.*, [12]). For example, the PRIME project [19] reports on several different paradigms for user management of an identity system, including the drag-and-drop actions we have incorporated. Our user interface was directly inspired by The Web Wallet [23], a tool for management of sensitive personal information on the web.

Usage Scenario

Suppose Sanjay is interested in shopping for holiday presents for his wife Kitty. He visits Amazon.com, but is having trouble finding the right book. Amazon suggests to Sanjay that it could recommend gifts based on what he and his wife's friends have been buying, if he is willing to share information about his friends. Amazon also reports that their privacy policy dictates that they will completely anonymize the data Sanjay sends, so his privacy is ensured.

Sanjay agrees and opens up the Databasket sidebar in his browser (Figure 1). He finds his collection of friends data (his 'Friends' basket), and drags it over to the Amazon.com web site. He drags the basket over a target on the web site; as he does, he is presented with a choice of sending this data only once, or permanently allowing Amazon to query his data. Deciding to share the data only once, he drops the basket onto the appropriate selection.

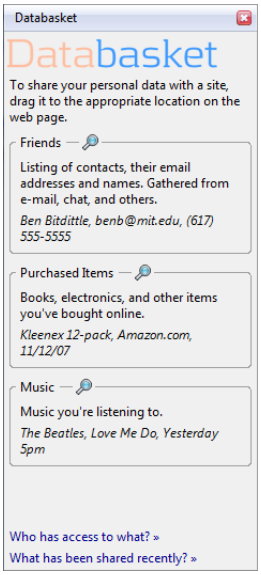

Figure 2: The Databasket sidebar interface in Firefox

The page reloads, and Sanjay notes that Kitty's friend recently purchased a particular fantasy book and rated it highly. Knowing his wife likes fantasy as well, he decides to purchase this recommendation. As Sanjay checks out, Amazon notes that Sanjay had previously elected to share his home address and credit card information with them permanently, and allows Sanjay to skip filling out that particular information. Amazon also notes that it is inserting the book into Sanjay's 'Media' basket, so other web sites can know he owns the book.

Meanwhile, Kitty is writing an email to Sanjay's coworker to decide what to get him for the holidays. She knows that she is Facebook friends with this coworker, but doesn't know his email address. Having shared her Friends information with Gmail, Kitty can type in his name like any contact and Gmail indexes into her Friends information in Databasket to import the contact.

The following section details the user interface that Sanjay and Kitty have used to enable this personalization.

User Interface

The user experience for Databasket centers around its privacy controls. Databasket does not share data with web sites unless the user grants explicit permission. Naive users are particularly given to simply 'clicking through' most warning messages without reading or understanding them, however, so it is of paramount importance that the users understand exactly what they are doing when they decide to share highly personal information with the web site. Researchers in usable security have developed a set of design guidelines for such systems (*e.g.*, [12]), and Databasket draws on these for design inspiration.

Granting Permission

The Databasket user interface is shown in Figure 2. It separates the user's personal data into a small number of baskets, including Friends and Contacts, Purchased Items, and Personal ID (a collection including his name, address and e-mail information). Each basket is draggable and can be received by specific Databasket elements on participating web pages. Web pages create drag targets advertising their use to the user, for instance an ability to receive Friends information. If the user wishes to share information with the web site, he or she grabs the basket in the Databasket sidebar and moves it over to the drop target on the web page. When the user enters the drop target, Databasket presents a larger target window to the user (Figure 3), asking the user to make a choice between sharing the data only once, or allowing the site permanent read permission to Friends data. By dropping the data on the appropriate choice, the user indicates his or her wish, and Databasket allows the web site read access to the data.

Sites may also write data to the Databasket central store, and users may wish to allow them write access in order to allow other sites to read the data. To grant write permissions, the process is reversed: the user drags an icon from the web site to the Databasket sidebar.

Examining Basket Contents

The user may click on the magnifying glass icon next to each basket's title to open up a Browser tab containing the contents of that basket (Figure 4). The data may be heterogeneous between and within baskets – for example, your music contains albums, tracks, artists, and possibly even music videos, some of which may be missing different fields – so we utilize Exhibit [16] for

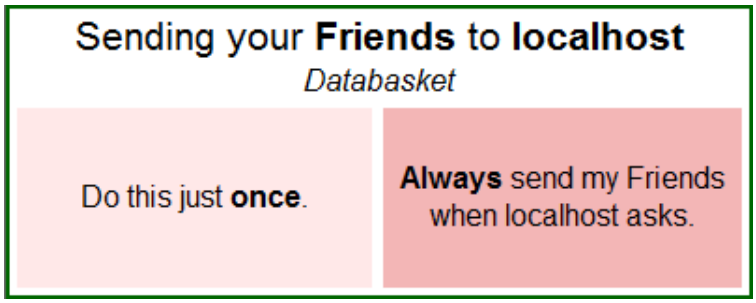

Figure 3: The Databasket drop target, injected onto the web page, asking users to differentiate between one-time and permanent permissions granting.

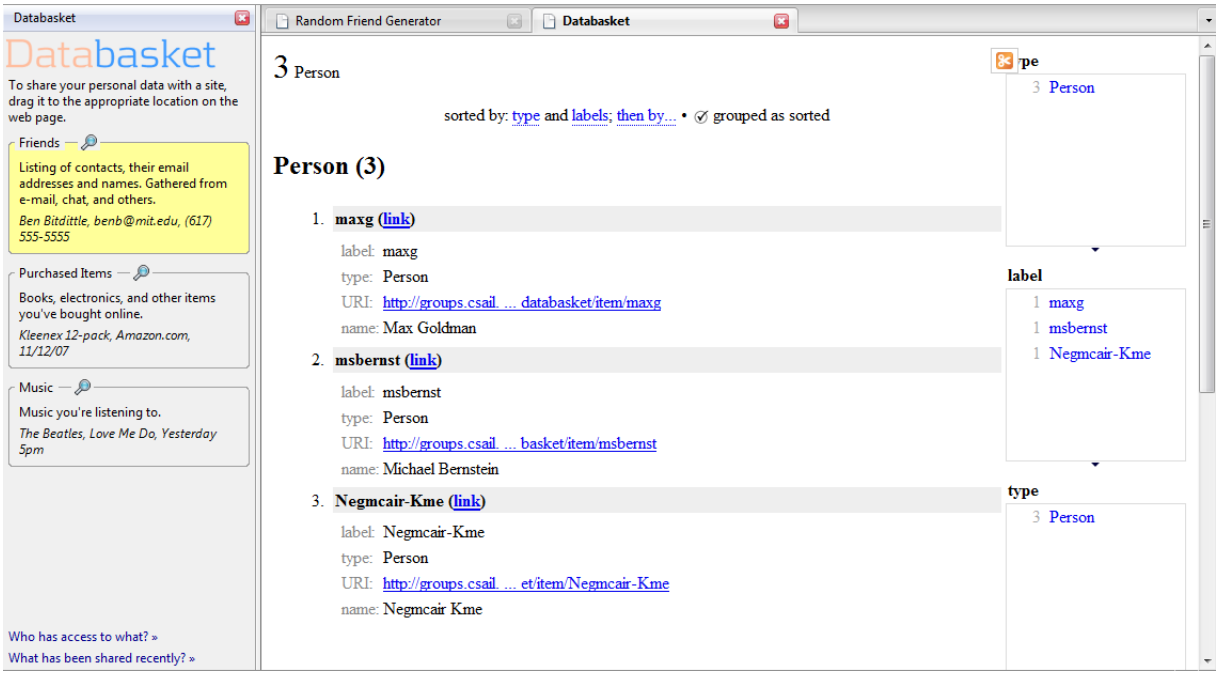

Figure 4: Examining the contents of the user's Friends basket using the Exhibit [16] interface.

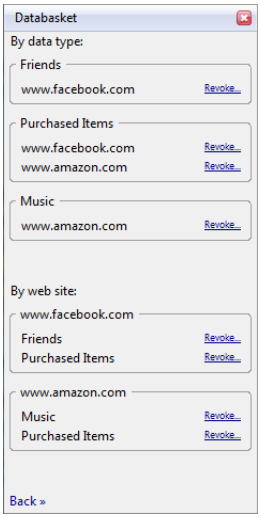

Figure 5: The permissions inspection interface, sorted at top by basket and at bottom by web site.

its ability to visualize and enable faceted browsing navigation across semi-structured data. Exhibit loads the data into a local web interface to allow the user to inspect the data being shared.

Inspecting and Revoking Permissions

Databasket allows the user to view currently shared data and revoke any privileges if desired. A link from the main databasket interface brings up the permissions inspection interface (Figure 5). This interface presents current permissions data with the user in two ways: indexed by basket (e.g., all web sites with read permission to the Purchased Items backet), and by web site (e.g., all baskets shared with Facebook.com). The user may use a "Revoke" link to deny the web site further access to that basket.

Auditing Personal Information Transfer (Not Yet Implemented)

A final part of the interaction (not yet part of our prototype) will allow the user to inspect the data that has been transferred to web sites. This historical record serves as a paper trail of sorts, allowing the user to understand what he or she has shared with other parties in the past.

Developer API

In addition to its user-facing components, Databasket presents an API to web application developers. Data in baskets are represented in semantic web format (RDF). Web applications access these data using query language SPARQL [7], and modify it with SPARQL/Update [8] (both standards proposed to the W3C). Both of these languages bear some resemblance to the SQL language for querying and updating relational data, but are designed to express relational constraints over a directed and potentially schema-free RDF graph.

To provide access to a user's data, Databasket injects a new global object, `databasket`, into the top-level of every web page's Javascript environment. In the current iteration, this object exposes three functions to web application developers:

`query(queryString, resultFormat)` This function executes the SPARQL query `queryString` in all baskets to which the site has access, and returns the results as a string in the specified `resultFormat`.

`update(basketUri, updateStmt)` Executes a SPARQL/Update statement `updateStmt` within the basket `basketUri`.

`addListener(listener)` Subscribes `listener` to permission changes for the page's domain; `listener` will receive a callback when this occurs.

This simple API is sufficient for web applications to perform queries and updates, and to react appropriately when their permissions are changed, for example by reloading their content or re-attempting some queries.

In order for developers to write appropriate SPARQL queries and updates, they must have some knowledge of the structure of the data within a basket. For example, FOAF is an RDF vocabulary for describing people [2], and it is reasonable to store contact information for friends in the 'Friends' basket as nodes of type `foaf:Person` with other FOAF-specified attributes. Such conventions (partially enforced by the system as described in the following section) would need to be learned by a new developer.

Implementation

All data in Databasket is stored in a persistent RDF store managed by Jena, a Java Semantic Web framework [5]. This includes both user data and data about baskets, permissions, and so forth. The Databasket backend uses

SPARQL and SPARQL/Update in preference to APIs provided by Jena wherever possible, increasing the flexibility of the system to changes in data model or RDF store implementation.

All data is stored together in one database, and the division of data into baskets is implemented by finding sub-graphs within the global graph that correspond to given baskets. Each basket is a node of RDF type `basket` that has some number of associated `basketTopLevel` RDF types. The basket also participates in a `basketItem` relationship with any number of other nodes whose type is one of the 'top level' types for this basket. Those nodes may have further connection to other nodes of other types.

Continuing the example from the previous section, the 'Friends' basket would have `foaf:Person` as one of its `basketTopLevel` types, and `basketItem` connections to several `foaf:Person`s representing the user's friends. The entire sub-graph of connected RDF nodes is considered to be part of the basket; this definition is straightforward and allows for simple sharing of data between baskets, but may need to be refined in the future.

Web applications are never given direct access to the `databasket` nodes in the store. In particular, while this is not a problem for executing queries, it means that an application cannot explicitly manipulate the necessary graph structure to add or remove items from a basket. Instead, the system requires specification of which basket should be affected by an update action, creates links to newly-added nodes of a `basketTopLevel` type for that basket, and removes links to deleted nodes.

The Databasket user interface is implemented as a Firefox web browser extension. The extension communicates with the Jena store through a Java remote interface, decoupling the data source and permitting future development of other data providers or consumers.

The Firefox user interface markup language has built-in support for using a template to generate user interface elements that display RDF data. We exploit this facility in order to dynamically construct the UI and automatically update it whenever, for example, the user grants or revokes permissions on a basket. The user interface itself uses a variety of Javascript techniques to implement direct manipulation.

In order to display the data in an Exhibit for users to inspect, the system simply makes a `SELECT` query to the appropriate basket, then translates the returned RDF-XML to Exhibit's JSON format by use of the Babel web API [1].

User Study

We conducted a formative evaluation of the Databasket user interface as part of the iterative design process. Test subjects were four graduate students who had previously been introduced to Databasket but had not interacted with it themselves.

In the evaluation, subjects used Firefox and Databasket initialized with three baskets, and only a few items in each basket. The goal of this evaluation was not to test whether the current interface could scale to many baskets with hundreds of items. Rather, we were interested in learning whether the general interface approach for permissions granting and monitoring made sense. Each subject was presented with a brief overview of Databasket and then began a sequence of tasks centered around manipulating their 'Friends' basket. The tasks were described to the user in concrete terms using an example friend and two web application mock-ups, but were roughly:

1. View data in a basket.
2. View permissions for a basket.
3. Grant read permissions for a basket.
4. Revoke read permissions for a basket.
5. Grant write permissions for a basket.

The evaluation uncovered a number of small usability problems, some of which were addressed between subjects. In general, users understood the purpose of Databasket and its interaction with web sites. All users had some initial difficulty with the metaphor of dragging data directly to the site to grant read access; in no case was this a user's first attempted action. More problematic was the metaphor of dragging from the site to the basket to grant write permission. Even with strong wording on the drag target ("drag me"), users still anticipated consistency with the read-granting interface. One subject pointed out that since write permission might include deletion as well as updating or adding data, the directionality implied by dragging from site to basket was not necessarily accurate.

Other common difficulties included mode errors with the baskets vs. permissions views; and questions about the granularity of permissions, both with respect to individual items within a basket, and individual pages or applications in a domain. Across all tasks, the cumulative success rate for

completion was 80%. All users completed all tasks after prompting from the experimenter on where to focus their attention.

Discussion

Our work faces two major challenges: mass adoption, and a challenge that it is opening up users to identity theft and unwittingly sharing more data than they would otherwise desire to. We address each in turn.

Adoption is a serious challenge for a tool like ours, as we require both data sources to fill the user's database of information and web sites willing to query it. Without enough users of the plug-in, major online businesses like Amazon and Facebook have no reason to share their data with the user, or to devote screen space to advertising their ability to take in data. However, without major web sites giving the user control over their data, there is nothing for the user to share with these sites. We propose to resolve this chicken/egg problem by seeding the user's database with information mined from their workstations via the PLUM [22] system. PLUM tracks users' email activity, windows and open programs, music playing, chat patterns, and much more. By importing some or all of this data, we have created an enticing opportunity for web sites to improve their personalization service. To make the service of immediate use to the user, we can also seed their database with personal information (name, contact info, etc.) that can act as a certificate and allow participating web sites to forgo or shorten login and registration procedures.

Databasket presents a veritable Pandora's Box of problems if misused. Users may unwittingly share data with malevolent web sites, thereby compromising themselves and very possibly their friends and contacts. Web sites may misuse the data, saving it and sharing it with other partners against the user's wishes. These violations of privacy could enable very effective security attacks on the user, as the phisher may use the person's identifying information to appear credible. We can offer a few modes of support. To avoid web sites inadvertently sharing personal data, we suggest that our model (the user owns the data, and the web sites query) allows the web site to maintain little or no personal information about the user that may be compromised. To address the question of keeping users from sharing data with phishing web sites, one might consider a signing procedure similar to current digital signature technology for payment interfaces, guaranteeing even-handedness. We can also maintain a blacklist of known phishing web sites. Databasket could turn the user interface a warning color if the user

seems intent on sharing data with either an unsigned web site or a known phisher. As a mediator of the data sharing process, Databasket could thus take an active role in ensuring the user does not take potentially dangerous actions.

Future Work

There are many avenues for future work with our efforts. Here we cover a few of interest: integrating the system with user modeling components to seed data, offering aggregated data to improve protection, and active information auditing as it is being shared.

The Databasket system can offer a much more compelling web experience if web sites are able to base personalization services on a wider variety of data sources. For example, Amazon could do a better job recommending books if it knew the authors were graduate students at MIT studying human-computer interaction, or that they listened to electronica music often while coding. Integrating with PLUM [22] would allow just such information to be placed into the Databasket framework; PLUM tracks all manner of user activity, including active windows, web sites visited, emails read, chats, music playing, and more. PLUM already uses a Jena semantic web format for encoding its observations, so integration will be straightforward other than mapping the data meaningfully into baskets.

Aggregating the user data presents a potentially useful way to preserve privacy for users while they share data. Sharing the distribution of friends living in particular areas, or the number of science fiction books owned (but not their titles), or the topics most salient in the user's e-mail conversations (but not the emails' contents themselves), may provide a middle ground between sharing potentially sensitive data and providing data of real use to web sites. Aggregates, however, complicate the user's mental model considerably: for example, is Databasket sharing friends' distribution by state, or county? Are we sharing the location distribution and last names, but not email addresses? Or email addresses, but not number of communications with each person in the last month? The challenge may lie in finding the appropriate balance.

Users' data are ever-changing, and they may forget with whom they have permanently shared data. As a result, users may not anticipate the results of this data flow between web applications, for example if Facebook automatically attempts to befriend a new Gmail contact with whom the user is having a flame war. Here, the user did not foresee the results of

her e-mail conversation as Gmail sharing the information with Databasket, and Databasket automatically forwarding this information on to Facebook. For this reason, it seems desirable to allow users to audit and potentially censor the data they are sharing. Databasket could support this in a number of ways; however, we do not wish to spam the user's attention with large numbers of dialogue boxes. Instead, we might consider a ticker tape or notification balloon style interface, sharing a subset of the information being sent each time it is about to be sent and giving the user a small time period during which he or she may intercept and cancel or amend the transmission.

Conclusion

Databasket allows semantic web data to be stored in a central, user-controlled repository and selectively shared with web applications. The Databasket interface within the user's web browser supports and promotes good decision-making when the consequences of sharing personal data could be difficult to reverse. By organizing data into understandable units – baskets – users gain broad power over their data-driven interactions with web applications. The Databasket vision is ultimately one in which web sites gain access to the information they need to provide accurate and compelling personalized services, and users regain control over the rich variety of data about themselves, to the mutual benefit of both.

References

[1] Babel. `http://simile.mit.edu/babel/`.

[2] FOAF vocabulary specification. `http://xmlns.com/foaf/spec/`.

[3] Google Gears. `http://gears.google.com/`.

[4] Greasemonkey. `http://www.greasespot.net/`.

[5] Jena semantic web framework. `http://jena.sourceforge.net`.

[6] OpenID. `http://openid.net/`.

[7] SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

[8] SPARQL/Update: a language for updating RDF graphs. `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`.

[9] R. Barrett, P. Maglio, and D. Kellem. How to personalize the Web. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 75–82, 1997.

[10] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[11] M. Bolin, P. Rha, and R. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172. ACM Press New York, NY, USA, 2005.

[12] L. Cranor and S. Garfinkel. *Security And Usability: Designing Secure Systems That People Can Use*. O'Reilly, 2005.

[13] J. Hong and J. Wong. Marmite: end-user programming for the web. *Conference on Human Factors in Computing Systems*, pages 1541–1546, 2006.

[14] D. Huynh, S. Mazzocchi, and D. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *ISWC*. Springer-Verlag GmbH, 2005.

[15] D. Huynh, R. Miller, and D. Karger. Enabling web browser to augment web sites' filtering and sorting functionality. In *Proceedings of the 19th Conference on User Interface Software and Technology (UIST)*, 2006.

[16] D. Huynh, R. Miller, and D. Karger. Exhibit: Lightweight structured data publishing. In *WWW*, 2007-05.

[17] D. Khushraj and O. Lassila. Ontological Approach to Generating Personalized User Interfaces for Web Services. *International Semantic Web Conference*, 2005, 2005.

[18] M. Krohn, A. Yip, M. Brodsky, R. Morris, and M. Walfish. A world wide web without walls. In *6th ACM Workshop on Hot Topics in Networking (Hotnets)*, Atlanta, GA, November 2007.

[19] J. Pettersson, S. Fischer-Hübner, N. Danielsson, J. Nilsson, M. Bergmann, S. Clauss, T. Kriegelstein, and H. Krasemann. Making PRIME usable. *Proceedings of the 2005 symposium on Usable privacy and security*, pages 53–64, 2005.

[20] D. Quan, D. Huynh, and D. Karger. Haystack: A Platform for Authoring End User Semantic Web Applications. *International Semantic Web Conference*, pages 738–753, 2003.

[21] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. *GroupLens: an open architecture for collaborative filtering of netnews*. ACM Press New York, NY, USA, 1994.

[22] M. Van Kleek and H. Shrobe. A Practical Activity Capture Framework for Personal, Lifetime User Modeling. *User Modeling*, 2007.

[23] M. Wu, R. Miller, and G. Little. Web Wallet: preventing phishing attacks by revealing user intentions. *Proceedings of the second symposium on Usable privacy and security*, pages 102–113, 2006.