Towards Enforceable Data-Driven Privacy Policies

Matthew Fredrikson, Drew Davidson, Somesh Jha
University of Wisconsin, Madison

Benjamin Livshits
Microsoft Research

Abstract

A defining characteristic of current web applications is that they are *personalized* according to the interests and preferences of individual users; popular examples are Google News and Amazon.com. While this paradigm shift is generally viewed as positive by both users and content providers, it introduces privacy concerns, as the data needed to drive this functionality is often considered private. Web applications have responded by giving users the chance to deny explicit disclosure of personal information, as well as minimizing the invasiveness of the information they require. In this position paper, we address the concern that explicit disclosure alone is not sufficient to protect user privacy, as attackers can combine users' consensually-shared information with additional background information to infer private facts about individuals. We argue that to properly account for these attacks, auditors must consider not just the relationship between disclosed information and attackers' background data, but also the semantics of applications that operate over the private information.

1 Introduction

Modern web applications are increasingly based on the idea that personalized functionality is both more agreeable to users and ultimately more effective than traditional static content. A handful of players such as Amazon.com and DoubleClick.com pioneered the use of personalized content on the web, and their success has led most other large, user-facing web services to follow suit.

While users have come to expect personalized functionality, they are also turned off by services that seem to violate their privacy. In an attempt to balance these conflicting needs, many services prompt the user for explicit consent prior to disclosure. Examples of this are Facebook's application platform, the privacy policies used on many websites, and recent systems discussed in the literature [6]. Many advertising networks now allow users to opt out of the default pervasive third-party tracking mechanisms used to target relevant advertisements, giving them the option of simply telling the network what their interests are. Finally, a growing number of browsers and services now support the "Do not track" tag, simultaneously raising awareness of privacy issues on the web and making explicit disclosure more pervasive yet.

We posit that explicit disclosure alone is not enough to ensure that a remote party cannot learn more about the user than she wishes. Given a small amount of seemingly innocuous information, a motivated attacker may still be able to infer more about the user than she is ultimately comfortable disclosing. Thus, we consider the problem posed by *inference attacks* on web applications, involving information from background as well as explicitly-disclosed sources (see Figure 1 for a graphical depiction). Our goal is to understand the properties of applications that allow one to reason about the information that can be leaked through inferences. Our key insight is that *precise conclusions about the inferences to which most web applications are vulnerable cannot be drawn without taking into account detailed information about the semantics of the applications themselves.* New strategies, analyses, and tools for dealing with this class of attack are needed.

In this paper, we examine several instances of this problem, discuss the analysis issues for precise privacy policy reasoning, and suggest a new notion of privacy that addresses the specific concerns of analyzing web applications that use personal information. Rather than superseding existing privacy definitions and mechanisms, our notion provides a way of incorporating them that is useful in analyzing the privacy of specific applications.

The rest of this paper is organized as follows: Section 2 discusses the privacy notion that characterizes our concerns. This section also includes three example scenarios in which such attacks are possible, and highlights the issues that call for further study. Section 3 discusses how existing notions of privacy play into this problem. We summarize our position and conclude in Section 4.

2 Towards a New Notion of Privacy

Consider a client-side application that uses Facebook's JavaScript Open Graph API to learn a user's birthday, gender, and zip code to provide localized coupon offers, personalized by gender. After learning about the user, the application asks a remote server for the set of coupons applicable to a given locale, passing the user's location as the only input to the remote server. Then, the application checks to see whether it is the user's birthday, and if so, presents any coupons that may be specialized for the occasion. Finally, the app displays the set of gender-specific coupons available to the user based on her location.

Despite the benign goal of this application, its privacy implications are severe. Researchers have found that 87% of United States residents can be uniquely identified by their birthdate, gender, and zip code [12]; it is no conincidence that this is precisely the information required by our example. However, as long as our application runs entirely on the user's client, the application developer will not necessarily be able to de-anonymize the user, because the only information leaked back to the developer is the user's zip code. The remaining information is used for local computations only, and then forgotten.

This exemplifies the need for enforcement techniques that incorporate program analysis to reduce the number of false positives encountered during privacy policy enforcement. In this example, an information flow analysis that determines which pieces of information from the query are communicated to an external host would suffice to certify the application free of any anonymity violations. Fine-grained, language-level information flow is a largely unsolved problem. While dynamic techniques for JavaScript have been proposed [3, 7], the associated overhead is prohibitive, and precision is an issue. Applying it to this problem will require further innovation and novelty.

A New Notion of Privacy. This example suggests a general way of thinking about the role of an attacker who wishes to violate the privacy of a user of such an application. Consider the following game: Alice has a secret predicate over a database of her personal information. If Eve guesses Alice's predicate correctly, then she wins the game. Our notion of privacy is concerned with Eve's probability of winning in two distinct "worlds" (depicted in Figure 1). In World A Eve has access *only* to a background database, which may or may not contain rows that pertain to Alice. In World B, Eve has access to a background database, a function, and the output of that function as computed over Alice's personal database. The notion of privacy relevant to our concerns in this paper seeks to characterize the difference in Eve's probability of winning between Worlds A and B.

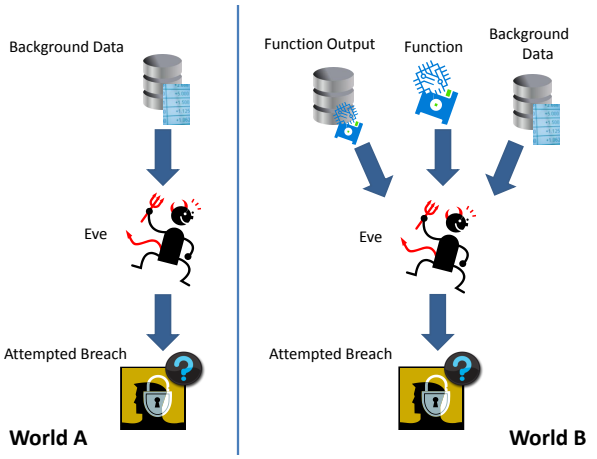

Figure 1: The differing types of information available to attacker Eve in worlds A and B reflect the privacy concerns posed by web applications.

Notice the correspondence between our previous example and this notion: Alice's personal database is her Facebook profile, the function is the coupon application, and the function output is the data communicated by the application to the remote server (i.e. Alice's zip code). Alice's secret predicate is her visible identity: the conjunction of her name and public attributes. If we attempt to understand the difference between Worlds A and B for this example by looking only at the information that the function inputs (i.e. birthday, location, gender), and the parties with which it communicates (i.e. an untrusted remote server), then we cannot conclude that this benign application is safe to run; program analysis is the only way to accurately characterize the difference in Eve's probability of winning between worlds A and B.

More Examples. A growing pool of applications draw personal information from multiple sources and eventually push portions of that information out to third parties. One example of this is LuckyCal [8], a personal assistant application that draws information from a number of sources (e.g. Facebook, Outlook, Google Calendar, Last.fm, etc.) on the web as well as the user's desktop to provide personalized event notification, scheduling, and booking. For example, if the user has marked a visit to New York City on the following weekend in her calendar, then LuckyCal will utilize the user's Facebook account to find friends in the area who might be free that weekend, and the user's Last.fm account to search for relevant live music events. LuckyCal will go as far as booking tickets, hotels, and other services on behalf of the user.

The privacy risks of LuckyCal are serious. Between calendar data, social networking data, and detailed records of interests in media and content, nearly all of the user's personal data is at stake. Furthermore, be-

cause LuckyCal uses all of this data to communicate with third parties on behalf of the user, the potential for unwanted disclosure is ever-present. For example, it is concievable that when searching for and booking tickets to events, LuckyCal could communicate enough information about the user's calendar to allow a third party to infer specific dates and locations of calendar entries. Alternatively, some of the data given to LuckyCal, such as the iTunes database with ratings, is "sparse", making deanonymization a possibility [11] if disclosure of such information is not done carefully.

In many ways, LuckyCal is a "killer app" for the personalized web: a personal assistant that automatically finds relevant events for the user based on a large, timely corpus of personal data. In all likelihood it poses no serious privacy threat, but the only way to safely and accurately determine this is to analyze LuckyCal's code: we must reason about the inferences that an attacker can make about LuckyCal's private inputs, based on its outputs. For example, we propose an analysis that determines whether unexpected discrepencies exist between the user's Facebook privacy settings and LuckyCal's public output. Traditional information flow analysis can provide a sound overapproximation of the information needed by such a procedure.

To address the concern that LuckyCal might leak the user's schedule and identity to a third party, we propose an analysis which determines that LuckyCal does not: *(1)* leak the precise date and location of a calendar entry, and *(2)* leak identifying information, such as interest data sparse enough to identify the user through external sources [11] (e.g. a Last.fm or Facebook profile page). One novel challenge with this analysis is determining the sparsity of the data released by LuckyCal; if this cannot be done accurately, then the analysis is likely to be too conservative for most users. In this case, traditional information flow is not appropriate, and new program analysis techniques are needed.

Transitioning to another example, recent work that demonstrates the effectiveness of "browser fingerprinting" [5] exposes another area in which program analysis-based reasoning about malicious inferences is needed. Nearly all modern browsers supply configuration information to sites for analytics and rendering. Eckersley showed that this configuration data can be used to form a "fingerprint" that uniquely identifies a particular browser [5]. His proof-of-concept fingerprinting tool, called Panopticlick, uniquely identifies 83% of all browser configurations, and 94% of configurations in which either Flash or Java is enabled. The effectiveness of this technique has important implications for privacy: it is a type of supercookie that can be used to track users without their consent.

Fingerprinting is difficult to evade because it relies on functionality that must be available to legitimate applications. Ad-hoc solutions are ineffective: Eckersley found that 99.1% of all attempts to alter a fingerprint by changing plugins or otherwise altering configuration could be thwarted, and the original fingerprint recovered. If a browser instead spoofs configuration values or simply disables "optional" components like JavaScript, then the page may not function correctly. Rather, we see potential to analyze the JavaScript contained on untrusted pages in order to determine the amount of identifying configuration information that is leaked outside of the browser. One analysis we propose uses information flow to refine the assessment of a page's fingerprinting threat: if information sources are invoked but not leaked off of the host, then there is no threat. Another promising direction that we will explore is to use Eckersley's information-theoretic assessment of privacy risk [5] in combination with a quantitative information flow analysis [9] to gain a more granular notion of leakage, thus reducing the risk of false positives further.

3 Existing Notions of Privacy

There is a large amount of previous work in database privacy upon which this work can build. However, as illustrated in the previous section, the problem we propose requires detailed reasoning about the semantics of web applications and the information that they make available to attackers. Here, we discuss some closely related work, and briefly consider the issues involved in adapting it.

Differential Privacy. Intuitively, differential privacy [4] ensures that the presence of a single individual in a database does not affect the outcome of a particular computation over that database. Usually, differential privacy is achieved by perturbing the results of computations performed over the database. Differential privacy is sure to play a role in providing privacy-preserving personalized content on the web; researchers have already applied it to related problems [10, 13], and for certain types of disclosure (e.g. aggregate statistics over sensitive data) it is the natural definition to apply. However, that an application claims to provide differential privacy is not sufficient: we seek procedures to verify that the actual implementation is free of selected malicious inferences. This might fail to hold because of a flawed or malicious implementation, or because a particular incarnation of differential privacy does not match the relevant privacy concerns. Chaudhuri et al. [1] have studied proofs of program *robustness*, or sensitivity of inputs to outputs, and briefly discuss a possible application to proving differential privacy. While we argue for analyses that verify more general notions of privacy than differential privacy, this might prove a worthwhile starting point for such an analysis.

Anonymity. There is a growing literature concerned with assuring anonymity throughout data disclosure. Work in k-anonymity [15] seeks to modify disclosed data so that any individual in the set cannot be distinguished from $k-1$ other individuals; as with differential privacy, there is potential future work in verifying applications that enforce this notion free of selected inference attacks. Recently, Narayanan and Shmatikov showed that on *sparse* datasets k-anonymity fails for a number of reasons [11], and it is usually possible to de-anonymize users even in the presence of perturbations. They quantify the amount of auxiliary information needed for their algorithm to de-anonymize a particular record in a given database, in terms of the number of necessary attribute values. This provides an excellent starting point for a program analysis that verifies anonymity, but issues remain, such as whether the analysis can be made general enough to account for attacks that use other algorithms.

Information Flow. There is a rich literature on the topic of information flow. Traditional notions of non-interference dominate most accounts (see Sabelfeld and Myers [14] for a nice survey of language-based techniques), and as discussed in Section 2, may play a role in reasoning about inference attacks. However, the binary "all-or-nothing" nature of strict non-interference is likely too restrictive for many applications; it may be necessary to reason more precisely about how the mutations to sensitive data before disclosure affect the inferencing ability of the attacker. For example, an application that computes and leaks an aggregate statistic over one's entire database of music ratings will look to a non-interference analysis as though it is leaking the full database, despite the fact that many useful statistics pose no privacy threat. Recent work in quantified information flow [9] eases the "binary" nature of non-interference, but it is not clear how to attach semantic meaning to a real-valued bit leakage measurement. Quantifying information flow based on an attacker's beliefs about a probability distribution [2] shows promise, but existing work is theoretical, and the matter of background information remains to be dealt with.

4 Conclusion

We have argued that it is not possible to come to safe and accurate conclusions about the inference-based privacy threats of personalized web applications without resorting to program analysis. A rich literature in database privacy [4, 10, 11] and information flow analysis [2, 14] provides a good starting point for developing new program analyses that address these challenges. In future work, we will explore new analysis frameworks, proof techniques, and decision procedures along these lines, as well as case studies on real applications. Our goal is to develop automatic techniques that lead to a precise, rigorous understanding of the threat posed by personal information disclosure on the web.

Acknowledgments. We would like to thank the reviewers and shepherd for their insightful comments and suggestions. We would also like to thank Bill Harris and Karl Voelker for their thoughts and suggestions on various drafts of this work. The first author of this work is supported by a Microsoft Research PhD fellowship.

References

[1] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust. Submitted, 2011.

[2] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17, October 2009.

[3] M. Dhawan and V. Ganapathy. Analyzing information flow in javascript-based browser extensions. In *Annual Computer Security Applications Conference*, December 2009.

[4] C. Dwork. Differential privacy: a survey of results. In *International Conference on Theory and Applications of Models of Computation*, May 2008.

[5] P. Eckersley. How Unique Is Your Web Browser? Technical report, Electronic Frontier Foundation, March 2009.

[6] M. Fredrikson and B. Livshits. RePriv: Re-imagining in-browser privacy. In *IEEE Symposium on Security and Privacy*, May 2011.

[7] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in javascript web applications. In *ACM conference on Computer and communications security*, October 2010.

[8] LuckyCal. `http://www.luckycal.com`.

[9] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *Conference on Programming Language Design and Implementation*, June 2008.

[10] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the net. In *International Conference on Knowledge Discovery and Data Mining*, June 2009.

[11] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, May 2008.

[12] P. Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57, August 2010.

[13] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Symposium on Networked Systems Design and Implementation*, June 2010.

[14] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21, 2003.

[15] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 10, October 2002.