Measuring and Predicting Web Login Safety

Xiao Sophia Wang
University of Washington
Seattle, Washington, USA
wangxiao@cs.washington.edu

David Choffnes
University of Washington
Seattle, Washington, USA
choffnes@cs.washington.edu

Patrick Gage Kelley
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
patrickgage@gmail.com

Ben Greenstein
Google
Seattle, Washington, USA
bengreenstein@google.com

David Wetherall
University of Washington
Seattle, Washington, USA
djw@cs.washington.edu

ABSTRACT

Users increasingly entrust websites with their personal and sensitive information. Sites commonly protect this information using user-supplied credentials (*i.e.*, logins). We conducted a measurement study of top websites and surprisingly found that they transmit these credentials in the clear, thus leaving them vulnerable to eavesdropping. To make matters worse, users are often unaware of this threat because sites and browsers reflect little information about how logins are handled.

As a first step towards solving this problem, we develop techniques for measuring logins on browsers to predict how logins would be handled *before* they are submitted. We demonstrate that achieving this goal requires instrumentation at the application layer and inside browsers. Specifically, network traces are not sufficient for determining login safety in general due to application-layer encryption; similarly, application-layer traces are insufficient because login submission logic may be generated in the browser at runtime. Based on a measurement study using login pages gathered from popular sites in addition to those visited by users through normal Web browsing, we found such predictions to be quite challenging due to a lack of any standard formats for Web logins. However, by applying a carefully chosen set of rules when measuring logins, we almost always correctly predict how logins will be handled.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Measurement

W-MUST'11, August 19, 2011, Toronto, Ontario, Canada.

Keywords

Web Logins, Encryption, HTTPS, Personal Information

1. INTRODUCTION

Internet users commonly entrust websites with personal and sensitive information pertaining to their identities, finances, and interests. To prevent other parties from accessing this data, websites typically employ login-based access control that entails sending credentials (usually a username and password) to a remote server for authentication. As such, login credentials are arguably the most sensitive information transferred on the Web and should receive strong protection. If compromised, all of the private data that they protect is at risk.

Despite the importance of website logins, their security has generally received little attention. We tested the top 100 websites in the United States according to Alexa [1], and found that credentials were exposed to eavesdroppers for fully one third of the sites. Surprisingly, a top site `wikipedia.org` and an insurance site `progressive.com` transmit logins in the clear. The threat posed by such exposure is real — it is trivial for an attacker with freely available traffic monitoring software to recover user credentials from any unencrypted Wi-Fi connection (e.g., at public hotspots such as airports and cafes). While the technical means to protect logins is clear – encrypting them with HTTPS – it is not widely deployed for a variety of operational reasons.

We argue that transparency is a critical missing feature for improving privacy [16]. Many studies show that websites send personal information to third parties without user awareness or consent. Users can choose to avoid such exposures if armed with knowledge about potential risks by simply not visiting those websites or visiting them on a secured network; further, making Web sites aware of such risks may encourage them to deploy secure authentication schemes. In the context of Web logins, our approach to enabling users to make informed decisions is to measure and predict Web login safety, then make this information available to users in an unobtrusive and intuitive way. Thus even without changing how and when passwords are encrypted, our approach can alleviate the risks of Web logins. Further, this approach only requires browsers to enable third-party code to observe how logins will be sent, enabling an immediately deployable implementation through browser extensions.

The goal of this work is to reduce this risk by notifying users about how their logins are protected *before* they are

sent. Unlike existing interfaces [3] that notify users when the *current* page has been loaded via an encrypted connection (e.g., the HTTPS padlock icon), our approach is to predict how and where upcoming login requests will be sent by measuring logins in Web pages. We argue that such information will help users make informed decisions about how they share credentials with sites they visit.

While the goal is straightforward, our measurements of Web logins reveal a number of interesting challenges. First, network traces cannot help here because we cannot find logins (to evaluate their security) in traces that are encrypted by higher level protocols; application-level traces also do not work because they do not reliably contain information about how logins are submitted. Second, there is a wide variety of non-standard login implementations because Web pages are constructed not only with HTML, but also with Turing complete languages (*e.g.*, Javascript), suggesting the need for flexible heuristics to identify logins. Third, we find that determining how login credentials are submitted is difficult due to the variety of technologies currently used for client-server communication. Last, our approach must run in real time while users interact with Web logins, so that we can notify them of how their logins would be transferred before they are submitted.

In this paper, we address these challenges by using a measurement study of 188 popular websites to develop a set of heuristics for determining Web login security. Our main contributions are as follows:

- We identify popular approaches to implementing Web logins, finding that there are more than six kinds of login buttons, only four of which can be identified directly. We also find four ways that logins are submitted, one of which enables perfect prediction of submission URLs at runtime.
- We use these results to build a Firefox extension that detects logins and predicts, in real time, whether logins will be protected during transmission. Based on an early deployment, we find that the rules for detecting login buttons work well for more than 93% sites; rules for predicting whether logins are sent encrypted and the top-level domains are nearly perfectly accurate.

The remainder of the paper is organized as follows. We discuss related work in the next section. In Section 3 we formally define the goals in this work and discuss the associated challenges. Section 4 presents the results from a measurement study to determine how popular Web sites implement logins and discusses the rules that we designed to identify and predict logins. In Section 5 we evaluate our rules. We discuss open issues in Section 6 and conclude in Section 7.

2. GOALS AND CHALLENGES

The goal of this work is to predict whether a Web login will be transmitted confidentially and to notify the user accordingly before its submission. At a high level, this requires first identifying the user interaction with a Web login, and then determining where and how the login information will be sent.

We define a Web login to be the username and password fields that are sent to a server for authentication. In particular, a login must contain at least one field that uses hidden characters (e.g., the HTML `password` type). Logins using third-party software (*e.g.*, Flash) are beyond the scope of this work. Fortunately, these are rare – of the top 100 sites, only one (`pandora.com`) adopts a Flash-based login.

After identifying a login, we predict its transmission safety, *i.e.*, whether it will be sent encrypted and whether the data is sent to the same domain as the Web page that contains it. This requires identifying the URL that the login is submitted to, *i.e.*, the submission URL. A submission URL contains a protocol, a domain and a port number (usually omitted). The protocol of this URL indicates whether a login is sent in the clear (HTTP) or encrypted (HTTPS), and the URL domain indicates the destination of a login.

The key challenges in this work are instrumenting end systems to identify Web logins and using this information to predict transmission safety in real time. The following sections address these issues.

2.1 Instrumenting Browsers

We argue that the most appropriate place to measure login encryption is inside Web browsers as users interact with them. Looking at HTTP requests when a request is made only indicates what is happening and cannot predict what occurs beforehand. Fortunately, HTML and Javascript code of Web pages reflects some information about future HTTP requests (*e.g.*, each embedded link implies a possible future HTTP request). Thus, measuring and analyzing Web pages is promising for predicting Web login safety. Also, analyzing Web pages from inside the browser application avoids the need to search packet payloads at the network layer for literal strings such as password fields and submission URLs. Conveniently, a modern browser engine constructs a Document Object Model (DOM) for each page, and all elements (*e.g.*, username and password boxes and login buttons) and their attributes are accessible programmatically from the DOM using Javascript. Further, the DOM includes information that is computed by the browser at runtime, allowing us to detect submission URLs that have been generated by Javascript at load time. Finally, the DOM is primarily intended for interactive use by websites and is thus an ideal platform for monitoring user interactions with Web logins.

Measuring at the DOM layer satisfies our constraints for enabling user notifications of Web login safety in real time and makes it trivial to identify key features such as password boxes. However, we find that identifying other key properties of Web logins such as login buttons and submission URLs is surprisingly challenging. Login forms and submission actions are often written in expressive languages like Javascript that can arbitrarily obscure the relationships between DOM elements (e.g., between a password field and a submit button), and can construct URLs dynamically, thus making them hard to predict.

2.2 Predicting Logins

A key challenge in this work is to display Web login safety information when a user is about to log in. Determining that a user is "about to" log in, however, is complicated by the fact that there are many ways that users can submit logins. For example, some users log in by typing in their username and password, then pressing the ENTER key. Others click the login button without typing in any login information if the browser stores passwords.

To cover these cases, we instrument two DOM events: focusing on the password box (users type in their passwords),

and hovering over the login button (users are about to log in by clicking on it). Again, the password box is easy to identify because of the standard HTML `password` type. Unfortunately, the login button is hard to identify because there is no HTML standard login button, and thus implementations differ. When looking at a collection of 100 popular websites (discussed further in Section 3), we observed six kinds of login buttons that are images or text, and that use a form, Javascript, or a link to submit logins. Given this flexibility, a solution must try to develop rules to recognize login buttons accurately.

Besides determining when a user is about to submit login information, we must determine where the login information is going to be sent. The submission URL dictates both that location and if it is sent with encryption. Unfortunately, the submission URL is not directly accessible from the DOM unless a page uses a form to submit logins. We find that many pages use an AJAX (asynchronous JavaScript and XML) request to submit logins where the submission URL could be assembled from several Javascript variables and is not directly detectable from the DOM. Worse, whether a page uses a form or an AJAX request is hard to detect from the DOM. In fact, we identified cases where a page has a submission URL in the detected login form but actually submits credentials via AJAX.

3. LOGIN DETECTION

This section discusses how we detect login buttons and submission URLs at the DOM layer. First we conduct a measurement study of the most popular websites to identify common implementations of login buttons and submission URLs. We generalize these results to develop heuristics for analyzing Web logins and incorporate them into a Firefox extension, Piigeon [15], that predicts login transmission safety and notifies users in real time.

3.1 Initial Web Login Study

To inform the design of Piigeon, we conducted a measurement study to understand how Web logins are implemented for popular sites. We first inspected 88 of the top 100 websites (12 sites have no logins) in the U.S. ranked by Alexa [1] as of March 17th, 2011. We refer to this data set as *DS-1*. To account for potential differences in Web login encryption when using less popular sites, we also look at 100 samples of the top 10,000 websites in the world ranked by Alexa [1] as of March 20th, 2011. We sample uniformly at every one hundred sites, and if a sample is not in English or provides no login, we keep iteratively selecting the next site on the list until we find a valid result. We refer to this data set as *DS-2*.

We manually inspect the login buttons of sites in DS-1 and DS-2 using Firebug [5], a Firefox extension that provides a UI for manual DOM inspection. To determine the submission URLs of logins, we inspect the `action` attribute of login forms. Finally, because we found that the `action` attribute is not always used for submission, we captured the actual submission URLs by inspecting HTTP requests and responses with the HttpFox [7] Firefox extension. Note that some sites have multiple logins; our current implementation detects at most one for each site.

Login buttons. Our measurements identify six DOM elements that implement the vast majority of login buttons (Table 1). We found that DOM elements `<input type='submit'>`, `<input type='image'>`, `<button type='submit'>`, and `<input type='button'>` always serve as login buttons if they are associated with a password in the same form. As a result, these types of login buttons, which account for more than 87% of websites in both datasets, are easy to identify with no false positives.

DOM Elements	DS-1	DS-2
`<input type='submit'>`	50%	62%
`<input type='image'>`	19%	23%
`<button type='submit'>`	18%	9%
`<input type='button'>`	0%	2%
`<* onclick='*'>`	7%	1%
`<a href='javascript:*'>`	1%	1%
Others	5%	2%

Table 1: Categories of DOM elements that implement the login button and their frequency. The vast majority of login buttons fall into six distinct categories, two of which use Javascript.

Methods	DS-1	DS-2
`<form action='url'>`, no js	80%	85%
`<form action='url'>`, with js	4%	4%
`<form action=''>`, with js	12%	8%
no form, with js	2%	2%
Others	1%	0%

Table 2: Categories of methods of implementing login submission URLs and their frequency. Only the first category is easily detected before credentials are submitted.

The other two element types embed nonstandard identifiers for login buttons. One uses the `<a>` element and incorporates the identifier in the `href='javascript:*'` attribute (we use `*` to represent an arbitrary string); the other uses the `onclick='*'` attribute. While the attributes could be used to make any DOM element serve as a login button, we have detected only `<a>` and `<img>` in our datasets. Despite the nonstandard tags, we observe common identifiers in the associated Javascript for these login buttons (e.g., "signin" and "logon") that enable us to develop heuristics to detect them.

Interestingly, 4% of sites in both DS-1 and DS-2 use elements that contain no identifiers indicating login buttons. In some cases, this is because the associated Javascript is located elsewhere in the DOM (e.g., in the header). We also find one site, `pandora.com`, that uses Flash to submit logins, which we cannot detect because the DOM does not provide access to Flash elements.

Submission URLs. To determine how sites implement submission URLs, we compare what we inspected in the DOM with the submission URLs actually used when the login is submitted. The four dominant methods for implementing the login submission URLs are listed in Table 2. In particular, we find more than 80% of the sites for both DS-1 and DS-2 use the `action` attribute of the `<form>` element to submit logins. This implies that we can correctly predict where the logins go and whether they are sent in the clear 80% of the time. We also find more than 10% of sites either have no associated form or have no `action` in their forms. Most of them use AJAX to submit logins, meaning the submission URLs cannot be detected from the DOM. Note that

when a form specifies no `action`, the result is that logins are submitted to the same URL containing the form.

We find that 4% sites use AJAX to submit logins even if they have specified a different URL in the `action` attribute, potentially leading to incorrect submission URL predictions. To determine the impact of such cases, we investigated further and determined that although the URLs from AJAX and `action` are different, the protocols and the top-level domains are still the same for each site. This implies that even when the prediction URL is incorrect, we are still likely to correctly predict whether credentials are encrypted and which domain they are sent to.

Summary. Our survey of Web logins demonstrates that the flexibility afforded to Web developers when designing sites leads to a variety of login buttons and login submission methods. For example, some sites might use an image or hyperlinked text instead of a standard login button. For login submission, we found sites using AJAX (e.g., to reduce wasted transmissions by verifying the login fields before sending) and some using Javascript outside of the login form (because JQuery [11] simplifies the code to do so).

Despite the variety of login buttons and submission methods, this initial study shows that rule-based detection is promising for identifying login elements. Our data shows that the vast majority of login buttons (between 92% and 98%) are identified by a simple rule, and the portion of logins not identified by these rules decreases with site popularity. We believe this occurs because less popular sites tend to use simpler ways to implement login elements. The next section discusses how we use these observations to detect logins at runtime.

3.2 Detecting Logins in Practice

This section discusses how we detect logins and predict login safety. Based on the measurement study in the previous section, we develop rules for detecting login buttons and predicting submission URLs. In this section, we describe the current set of rules; we evaluate its effectiveness in the next section.

To experiment with these rules in the wide area, we incorporated them into a Firefox extension called Piigeon [15] and made it publicly available. Piigeon presents a red (unsafe) or green (safe) indicator only when users type in passwords or hover over login buttons. Figure 1 shows an example for an unsafe login form.

Identifying login buttons. When a page is loaded, our tool identifies candidate login buttons by searching the DOM for elements that match any categories in Table 1. For the first four categories we use simple matching, as they have no false positives according to our measurements. For categories that use Javascript in login buttons, we use heuristics to identify candidates. In particular, we exclude candidates with keywords such as "forgot" and "keep" in their text attributes to eliminate false positives such as a link for "forgot your password" and a checkbox for "keep me logged in". We only include candidates with keywords such as "signin" and "logon" in their Javascript attributes.

After determining the candidates, we further check whether elements are related to a login. We look for their ancestors in the DOM tree until we find a `<form>` element that contains a password box. This excludes a variety of DOM elements such as search buttons.

Figure 1: An example of Piigeon predicting an unencrypted login. When users type in their passwords, Piigeon displays a red icon (circled) to indicate that tumblr.com sends a login in the clear.

We currently do not support login buttons listed as 'Others' in Table 1 because there are no identifiers indicating login buttons. Similarly, our rules do not work for login buttons that have no form in their ancestors.

Predicting submission URLs. We now describe how Piigeon detects login submission URLs. After identifying a login button and a password box in a form, Piigeon checks the `action` attribute of the form in the DOM. Because the `action` attribute is always an absolute URL in the DOM regardless of whether it is absolute or relative in the HTML code, this provides a straightforward prediction for the submission URL. Note, however, that this could lead to incorrect predictions when a page uses AJAX to log in but also specifies a URL in the `action` attribute.

Under certain conditions, our rules are unable to detect logins that are not specified in the `action` attribute and that have no form. To this end, Piigeon leverages historical information recorded by listening to the HTTP traffic to determine the actual submission URL. The next time a user visits the same login, Piigeon predicts this URL. Although this approach fails to reflect live information, it reflects what happened in practice. As sites rarely change login code frequently, this approach should work well in practice.

Note that sometimes we cannot detect logins that are submitted using `action`. This only occurs for logins that do not specify in the `action` attribute and submit directly to their page URLs. We cannot distinguish between them and logins that use AJAX.

4. EVALUATION

This section evaluates our rules to detect login buttons and submission URLs. To achieve this, Piigeon reports anonymized detection and prediction statistics to a secure data-collection server. We use data gathered from beta testers from March 10th to June 12, 2011, consisting of 937 distinct retrievable logins in total. The data directly provides the accuracy of predicting submission URLs; for determining the accuracy of detecting login buttons we manually inspected the associated Web pages.

Table 3 shows the type and frequency of login buttons based on user behavior. Note that the distributions are similar to those in Table 1, suggesting that our measurement study formed a representative set of sites.

DOM Elements	Comp.	FP	FN
`<input type='submit'>`	536 (57.2%)	0	12
`<input type='image'>`	210 (22.4%)	0	5
`<button type='submit'>`	109 (11.6%)	0	1
`<input type='button'>`	4 (0.4%)	0	0
js, `<a href='javascript:*'>`	29 (3.1%)	0	19
js, `<* onclick='*'>`	32 (3.4%)	0	12
Others	17 (1.8%)	0	17
Total	**937**	**0**	**65**

Table 3: Frequency of login buttons types from sites that users visited. In the current deployment, our rules have had no false positives (FP) and a relatively small number of false negatives (FN).

Ways	Comp.	E_P	E_D	E_{TD}
`<form action='url'>`, no js	837 (89.3%)	0	0	0
`<form action='url'>`, with js	42 (4.5%)	4	8	0
`<form action=''>`, with js	55 (5.9%)	-	-	-
no form, with js	3 (0.4%)	-	-	-
Total	**937**	**4**	**8**	**0**

Table 4: Frequency of login submission methods for sites that users visited, along with the number of erroneous predictions. E_P, E_D, and E_{TD} represent incorrect predictions for protocol, domain, and top-level domain, respectively. Our rules never incorrectly predict that a login will be submitted with encryption.

Table 3 also shows the results from evaluating the false positive (FP) and false negative (FN) rates for of our rules. A false positive occurs when the detected element is not the actual login button; a false negative occurs when the login button is not identified. Fortunately, our rules generate no FPs which means that the heuristics to exclude elements other than login buttons are (so far) resilient in practice. Note, however, that our rules generate 65 (6.9%) FNs. When identifying the root causes, we find that 17 FNs are from login buttons that embed no identifiers of logins in their attributes – a category that is not currently supported by our tool. Of the 31 FNs based on the `<a href='javascript:*'>` and the `<* onclick='*'>` type, we find that 18 can be captured by extending existing heuristics. This result suggests the importance of incorporating user feedback to improve the coverage of our tool. The remaining 17 FNs are caused by other reasons (*e.g.*, user interactions with a site cause the DOM to change, but Piigeon is not notified) and are not related to our rules.

We now focus on submission URL prediction. Table 4 shows the distribution of login submission methods. We find 837 (89.3%) sites use an `action` attribute to submit logins – in these cases Piigeon always correctly predict the submission URL.

The remaining cases consist of 55 (5.9%) sites that do not specify an `action` attribute in their login forms, and 3 sites do not embed logins in a form. For such sites, our rules will not predict the submission URL the first time the login is encountered, but it will subsequently correctly predict the URL using historical information. Interestingly, we find 42 (4.5%) sites use AJAX to submit logins but still specify a URL (unused by the browser) in the `action` attribute.

We further evaluated our rules according to the accuracy of predicting login transmission encryption. Since this is indicated by the protocols used in the submission URLs, we focus only on the 42 logins that we predict the submission URLs incorrectly. We found only 4 encrypted logins (*i.e.*, `xpressengine.com`, `expressscripts.com`, and two in `pse.com`) that was reported as unencrypted, yielding an error rate of 0.4% logins. By inspecting this login, we find that `xpressengine.com` specifies `action='./'` and `expressscripts.com` specifies `action='#'` which correspond to an empty `action`. We have already modified Piigeon to incorporate this behavior.

Finally, we evaluate our rules according to predictions of which domains logins are sent to. While we predict eight logins incorrectly in terms of the login submission URLs, we found that all of these cases were still sent to the same top-level domains as the predicted ones. Thus, even when our rules do not work as intended, they do not incorrectly predict the safety of Web logins.

5. DISCUSSION

Our results indicate that it is feasible to use in-browser measurements to predict Web login safety and reflect this information to users in real time. While we were able to achieve good results for popular websites, there remain a number of open issues.

For one, manually inspecting and identifying Web logins for incorporating into our rule set simply does not scale. To address this limitation, we are currently experimenting with a crowd-sourcing approach that relies on information about Web logins collected from Piigeon. Our goal is to use this information to further extend our current rule set, then deploy these rules to users. An interesting area of future work is determining how to automatically generate and share these rules, *e.g.*, using heuristic, statistical and/or machine learning approaches. We do not plan to use taint analysis, not only because it will introduce high overheads that may not work in real time, but also because literal search works well for tracking passwords.

While crowd-sourcing is promising, there remain some Web logins that our tool cannot identify and thus cannot be collected from users. One potential solution to this problem is to extend the DOM to make identifying Web logins easier. In particular, if the DOM were to tie HTTP requests to information from password boxes, the login submission problem would become trivial. The drawback of this approach is that it requires browser kernel modifications and is thus difficult to deploy broadly and uniformly across browsers.

Another issue is that users have limited options when notified about an unsafe Web login – they can leave the website or submit logins despite the risk. To address this, we plan to automatically detect whether sites support HTTPS and offer users to switch to HTTPS. We also plan to make our ongoing results publicly available through an interactive website, both to provide motivation for unsafe sites to improve how they protect Web logins and to enable non-Piigeon users to make informed choices about which sites to use.

Finally, we plan to extend our approach to the more general problem of privacy in Web interactions. One of the key challenges is identifying private information (*e.g.*, names, addresses and phone numbers) outside the limited scope of logins. Another important research direction is moving be-

yond the Web client side of interactions and inferring how private information is stored and shared among sites.

6. RELATED WORK

Our work is closely related to approaches that improve the transparency of how private data is shared in general, and to approaches that attempt to enforce secure Web browsing.

Informing users of their privacy. Informing users of their exposed privacy is an alternative (to technical solutions, *e.g.*, encryption) that can improve privacy [16]. Unfortunately, both goals (*i.e.*, informing users and reflecting privacy in practice) are rarely met at the same time. Most sites provide users with privacy policies that are often difficult to read [14, 13] and sometimes even incorrect. Browsers provide padlocks and other cues to indicate page encryption. However, these cues are often ignored or misunderstood [17].

The Web of Trust (WOT) [18] project solicits privacy and trustworthiness ratings of websites from users and reflect this information to users as they browse websites. However, a user's trust in WOT is only as strong as their trust in the users – potentially non-experts – who rate sites. Other work measures the private information flows in practice without reflecting them to users in real time [12, 2, 10]. In addition, these approaches do not predict the privacy of future actions and thus is not sufficient to inform users how their data will be protected before it is transmitted.

To reflect exposed privacy in practice, we argued that there should be low-level support to measure and reflect privacy on Web and mobile applications [16]. This work is a first step towards both measuring and reflecting a piece of private information (whether logins are sent encrypted) in practice. By predicting this information beforehand, our work enables users to take action to protect their own privacy before exposing information.

Adopting HTTPS. A common approach to providing privacy for Web interactions is to encrypt all transmissions between a client and a server using HTTPS. At a minimum, Web traffic related to the most sensitive information (*e.g.*, logins and financial information) should be transmitted using HTTPS. Unfortunately, while all moderns browsers support HTTPS, the protocol requires support at both endpoints and thus the security of Web site interaction largely depends on Web sites. A number of solutions attempt to encourage or enforce encrypted Web interaction. One approach is to exert pressure on websites to support HTTPS, *e.g.*the Electronic Frontier Foundation (EFF) [4] efforts surrounding the Firesheep [6] extension. Another solution is enforcing HTTPS on browsers, based on the hypothesis that many websites support encryption but use HTTP by default. In this vein, the HTTPS-Everywhere [8] extension manually inspects a list of pages/requests that support encryption and enforces HTTPS for those pages. This approach is not feasible at large scale and is not robust to website changes over time. A similar extension, HTTPS-Finder [9], automatically detects pages that support encryption and enforces HTTPS on those pages. However, it only enforces HTTPS of loaded Web pages and does not apply to data transmitted to Web servers (*e.g.*credentials).

7. CONCLUSION

This paper demonstrates an important first step toward improving the privacy of Web interactions by informing users whether their Web login credentials are protected before they are transmitted to a server. We used a study of 188 popular Web sites to demonstrate that it is possible to predict, in real time, whether Web logins are transmitted with encryption and where this information will be sent. By making this information available to users when they enter login information, we enable informed decisions about whether to submit login credentials. We implemented this approach as a Firefox extension and made it available to users. Based on results gathered from beta testing, we show that our software can accurately identify 93% of logins and almost always correctly predicts whether login credentails are transmitted securely.

8. REFERENCES

[1] Alexa - the web information company. `http://www.alexa.com/topsites/countries/US`.

[2] K. Borders and A. Prakash. Quantifying information leaks in outbound web traffic. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009.

[3] S. Consolvo, J. Jung, D. Avrahami, P. Powledge, B. Greenstein, and G. Maganis. The wi-fi privacy ticker: Improving awareness and control of personal information exposure on wi-fi. In *Proc. Ubicomp*, 2010.

[4] Electronic frontier foundation. `https://www.eff.org/`.

[5] Firebug. `http://www.getfirebug.com`.

[6] Firesheep. `http://codebutler.com/firesheep`.

[7] Httpfox - an http analyzer addon for firefox. `http://code.google.com/p/httpfox/`.

[8] Https everywhere. `https://www.eff.org/https-everywhere`.

[9] Https finder. `https://code.google.com/p/https-finder/`.

[10] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in javascript web applications. In *Proc. CCS*, 2010.

[11] Jquery. `https://www.jquery.com`.

[12] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proc. CCS*, 2008.

[13] P. Leon, L. Cranor, A. McDonald, and R. McGuire. Token attempt: The misrepresentation of website privacy policies through the misuse of p3p compact policy tokens. In *Proc. PETS*, 2010.

[14] A. McDonald, R. Reeder, P. Kelley, and L. Cranor. A comparative study of online privacy policies and formats. In *Proc. PETS*, 2009.

[15] Piigeon. `http://www.piigeon.org`.

[16] D. Wetherall, D. Choffnes, B. Greenstein, S. Han, P. Hornyack, J. Jung, S. Schechter, and X. Wang. Privacy revelations for web and mobile apps. In *Proc. HotOS XIII*, 2011.

[17] T. Whalen and K. M. Inkpen. Gathering evidence: use of visual security cues in web browsers. In *Proc. Graphics Interfaces*, 2005.

[18] Wot - web of trust. `http://www.mywot.com`.