Sensitive Data Requests: Do Sites Ask Correctly?

Craig A. Shue and Minaxi Gupta
Computer Science Department
Indiana University
{cshue, minaxi}@cs.indiana.edu

Abstract— **To ensure the security of sensitive Web content, an organization must use TLS and do so correctly. However, little is known about how TLS is actually used on the Web. In this work, we perform large-scale Internet-wide measurements to determine if Web sites use TLS when needed and when they do, if they use it correctly. We find hundreds of thousands of pages where TLS is either not used when it should be or is used improperly, putting sensitive data at risk.**

I. Introduction

The Web offers unprecedented opportunities for e-commerce. The security of such transactions is commonly provided through the use of the Transport Layer Security (TLS) protocol [1], the standards track successor of the Secure Sockets Layer (SSL) protocol. TLS allows clients to verify the authenticity of the servers they access and ensures the confidentiality of the communication between the client and server. While previous work has analyzed TLS certificates and the protocol itself, little work has focused on its Web usage.

This paper is motivated by the desire to learn how TLS is being used on the Web today. A large portion of Web content is publicly available and does not require confidentiality. In many cases, such as reading news articles or using search engines, the benefits of TLS protection do not outweigh the performance overheads associated with the protocol. In other cases, sensitive information is transmitted and should be protected by TLS. However, simply using TLS is not enough; it must still be used correctly. To investigate TLS usage on the Web, we ask two primary questions: *Are there sites on the Web that are not using TLS when they should? Do the sites that use TLS do so correctly?* The motivation for the first question is that sensitive information can be easily intercepted by an eavesdropper unless TLS is used. The second question is motivated by the observation that TLS protection must begin *before* the Web server transmits a form to the client. Otherwise, a page containing the form can be altered by an attacker, allowing the interception of sensitive data. Several large organizations, including `amazon.com`, `chase.com`, and `hotmail.com`, have established the TLS protection after the client has downloaded the page, but before submitting the form data. This practice, called *secure post*, is typically used by organizations which have a high volume of user traffic that never signs into a form on the page. This is particularly common when the form appears on the main page of a site. These organizations use secure post to avoid the performance overheads associated with TLS for the non-authenticating clients. Unfortunately, this practice provides an opening for attackers to impersonate the Web sites and launch a man-in-the-middle attack on the Web clients.

To examine the extent of these poor security practices, we implemented a Web crawler and examined HTML forms on 4.3 million Web pages. We made several key observations from this analysis. First, 31-36% of Web pages do not use TLS at all when they should. To address this issue, we have implemented a browser extension that warns users about entering SSNs and credit card numbers on Web pages that do not use TLS in addition to identifying fields asking sensitive data. This results in fewer, but more accurate warnings. In manually evaluating the effectiveness of the extension, we found no false positives and two possible false negatives. Second, we find that of insecure pages that had forms, 1.65%-4.49% had at least one form that was submitted via HTTPS, resulting in the secure post vulnerability. If exploited, the insecure entry points can lead to fraud, possibly with significant financial repercussions for the users and the vulnerable sites. We propose a browser extension that attempts to verify these entry points using TLS before submitting sensitive data and issue a warning if such verification fails.

The rest of this paper is structured as follows. In Section II, we discuss our data collection and methodology. In Section III, we examine sites that offer no TLS protection with sensitive data and suggest some precautions users can take. In Section IV, we analyze sites that misuse TLS and suggest client-based strategies to correct the problem. We review related work in Section V and conclude in Section VI.

II. Data Collection and Methodology

To gain insights on TLS usage, we performed large-scale, Internet-wide Web crawls. We divided our crawls into four data sets, which were selected to capture different types of Web pages: popular pages, those visited by machines on our network, and those selected randomly.

In the first data set, which we refer to as the `DMOZ Breadth` data set, we obtained a list of URLs from the DMOZ Open Directory Project [2]. The DMOZ project consists of user-submitted links that form a directory for finding data, rather than using a searching approach. The data set, collected on February 13, 2008, contained 9,165,163 links. Of these, 4,025,911 links were unique. Most of these links used HTTP, not HTTPS, implying that they did not use TLS. A total of 2,312 links used TLS. We eliminated these TLS-protected links from further consideration since any forms on these pages will be transmitted securely by default. Over the course

TABLE I

OVERVIEW OF DATA SETS

Data Set	Total Pages	Pages with Forms	Total Forms	Insecure Forms	Secure Forms
DMOZ Breadth	3,213,764	692,869 (21.55%)	1,710,819	1,656,047 (99.67%)	54,772 (0.33%)
DMOZ Depth	78,726	37,879 (48.11%)	66,506	64,213 (96.55%)	2,293 (3.45%)
Alexa	344,868	269,600 (78.17%)	702,325	674,831 (96.09%)	27,494 (3.91%)
DNS	642,013	448,015 (69.78%)	967,990	938,555 (96.96%)	29,435 (3.04%)

of a few weeks, we were able to retrieve a total of 3,213,764 pages from the DMOZ links. This breadth-based crawl was superficial; it only examined the page directly linked from DMOZ. While this strategy allowed our crawler to examine pages from a large number of domains, it would not capture forms on secondary pages.

For the remaining data sets, we performed more detailed crawling. For each of these data sets, we obtained a URL for a top page, downloaded that page and any page linked from that page that was within the same DNS domain as the original page. This more detailed crawling limits the breadth of domains, while finding forms that are directly linked from the main page. Some URLs may be present in multiple data sets. Due to its unique crawling methodology, we allow the `DMOZ Breadth` data set to overlap with the remaining three without attempting to eliminate overlaps.

The second data set (`DMOZ Depth`) again uses links from the DMOZ Open Directory Project. However, rather than conduct a full sweep, we randomly selected $16,500$ unique links to perform our crawl. This allows us to directly compare the strategy of superficial crawls verses detailed crawls in finding forms. We obtained $78,726$ Web pages from this crawl.

In our third data set (`Alexa`), we analyzed popular Web sites. We used the Alexa Web Information Service [3], which ranks the most popular Web sites on the Internet, to obtain the $1,000$ most popular sites in each of 16 top level categories, as well as the top 500 most popular sites overall. Some sites were present in multiple categories; upon removing duplicates, we found $15,341$ unique Web sites. We used each of the sites obtained from Alexa as starting pages for Web crawling. This crawl resulted in $344,868$ Web pages.

In the final data set (`DNS`), we focused on actual user behavior. To create this data set, we captured all the DNS queries issued on our departmental network for a one-week period. We used the host names contained in the `A` (Address) record queries as the base for URLs for Web crawling. This data contained $164,145$ unique host names. From this crawl, we obtained $642,013$ Web pages.

For each data set, we parsed the HTML code of each page we downloaded. We used the `form` HTML tag to identify requests for data. For each form, we extracted the address of the page serving the form, the destination of the form, as well as each of the associated input fields. From this data, we could characterize the type of data being transmitted and whether the data requested from the user would be transmitted securely.

We examined 4.3 million Web pages. Not all the pages we examined contained forms. However, many pages contained multiple forms, as shown in Table I. For example, in the `DMOZ Breadth` data set, we found that a total of 692,869 (21.55%) pages contained a total of 1,710,819 forms. Using the `action` attribute in each `form` tag, we inferred if TLS was being used by looking for the presence of HTTPS. Otherwise, we inferred that the form was transmitted insecurely. In each data set, over 95% of the pages with forms contained only HTTP (insecure) forms. A relatively small number, 1.65%-4.49%, contained only forms submitted via HTTPS. We note that popular pages, those in the `Alexa` data set, had a higher percentage of pages with forms and a higher number of forms per page. Less popular pages had far lower rates of form usage. *Collectively, our data contained* $3,333,646$ *(96.69%) insecure forms and* $113,994$ *(3.31%) secure forms.* In the next two sections, we examine the insecure and secure forms separately.

III. TLS IN SENSITIVE DATA REQUESTS

The first question we sought to answer was: *Are there sites on the Web that are not using TLS when they should?* We analyzed all the 3.33 million forms transmitted insecurely toward this goal. To characterize the type of data transmitted by the form, we inferred usage from the `name` attribute on each form `input` tag as well as the `type` attribute of the HTML tag being used. For example, in the HTML code example below, the third line contains an `input` tag with the `type` attribute of "password," which is likely to be considered sensitive to the Web user. We note that lines 3 and 5, "Username:" and "Password:" are simply labels for the end-user to know what data to enter in the field; for technical reasons, these are not currently included in our heuristics.

```
<form action="http://www.example.com"
      method="post">
   Username: <input type="text"
                    name="user"><br>
   Password: <input type="password"
                    name="pass"><br>
   <input type="submit" name="page"
          value="Login">
</form>
```

We applied two simple heuristics to classify material as sensitive. In the first, we examined whether any `input` HTML fields were of `type` "password." The other heuristic was to examine the `name` attribute of any form-related tags in order to infer their usage. For example, `input` fields with "user" as the `name` attribute could be inferred to stand for "username," a piece of data we consider sensitive. Fields with "query" or "search" as the `name` attribute are more likely to be involved in Web searches; we do not consider this data to be sensitive. We develop patterns to match the value associated with `name` attribute in form fields to determine the data requested. We classify data that pertains to a user's identity or accounts as sensitive. *Accordingly, we consider user names, passwords, account numbers, addresses, (credit) card numbers, email addresses, real names, cities, and phone numbers to be sensitive data*. Some of this information is considered to be more sensitive than other data. For example, credit card information is likely to be more sensitive than an email address. However, an email address is tied to a user's identity and could be used for tracking purposes. Some may consider their home city to be sensitive data while others may not. In this analysis, we focus on a few pieces of sensitive data to determine the number of requests for that data rather than create an exhaustive set of sensitive data.

In Table II, we list the categories of sensitive data requested by insecure forms. Each `input` field is categorized exclusively through a series of rules. For example, a field with "password" as the `type` attribute is classified only as a "password field," regardless of any other matches on the `name` attribute. However, a page can have multiple categories of sensitive form fields, in which case it is counted under each type. *Overall, we found that 31.39%-36.00% of pages with insecure forms contained at least one sensitive field*. Specifically, we found that over 240,000 Web pages contain insecure forms that have `input` tags with "password" for the `type` attribute. Each of these password fields cause Web browsers to obscure the text entered into the field, to prevent others from seeing the data entered on the user's screen. Clearly, the Web sites consider this data to be somewhat sensitive, yet do not provide real protection for it. For the rest of the field categories listed in Table II, we searched for patterns in the `name` attribute of the `input` tag to infer the type of data requested. For example, "account" and "acct" are both substrings that we classify as requesting "account" information. The most commonly inferred requests were for email addresses, in which over 215,000 pages requested this information. As one may expect, data that is of extreme sensitivity, such as account and card numbers, is seen less frequently, appearing on less than 1,100 pages. While some categories may swap places in these data sets, they typically do so with neighboring categories. We do not notice any discernable correlation between popularity and type of sensitive information requested.

A. Browser Extension to Help Users Avoid Sending Sensitive Data through Insecure Forms

While Web servers may request that users submit sensitive data through insecure forms, Web browsers can advise their users about the risks of doing so. To some extent, such a protection already exists in many popular Web browsers: they warn users whenever they send any form data through an insecure medium. Unfortunately, the power of these warnings is diluted by the fact that they are indiscriminate. For example, being required to acknowledge the insecurity of each Web search performed would be an annoyance to most users while simultaneously encouraging users to always ignore such warnings.

TABLE II
THE NUMBER OF PAGES WITH DIFFERENT CATEGORIES OF SENSITIVE DATA REQUESTED BY INSECURE WEB FORMS

Field Type	DMOZ Breadth	DMOZ Depth	Alexa	DNS
Password Fields	120,541	4,645	51,907	66,622
Inferred Email	97,413	6,356	40,127	72,204
Inferred User Name	75,262	2,965	40,383	45,283
Inferred Real Name	60,230	3,170	19,791	40,936
Inferred Address	21,180	994	5,785	10,305
Inferred Phone Number	7,278	714	1,979	4,614
Inferred City	6,605	492	3,316	5,781
Inferred Password	3,073	82	1,085	1,290
Inferred Account	477	16	235	353
Inferred Card Number	282	27	230	155

Our solution is to use more fine-grained warnings. By eliminating overly broad warnings, users may be more likely to heed the warnings that remain. *We implemented and released an extension to the Mozilla Firefox Web browser that performs heuristics on the forms completed by the user and the data that they entered*[1]. The extension examines each form field to determine if sensitive data is being communicated. If so, the extension displays a warning to the end-user. For example, if a form uses an `input` tag with "password" as the `type`, the extension will create a targeted alert message if the form is being submitted without TLS protection. In the alert, the extension describes the sensitive information being entered to encourage users to consider what information they are submitting. As with our heuristics for finding sensitive data, the extension also searches for patterns matching sensitive data in the value of the `name` attribute for each `input` tag in the HTML document. We use the same patterns of sensitive data described in the previous section.

Our browser extension goes beyond examining the HTML document to find sensitive fields. It also examines the user's input to determine what type of information is being entered. We have implemented two heuristics for this: one to detect United States Social Security Numbers (SSNs) and one to detect credit card numbers. If the user enters a nine digit number separated into a sequence of three digits, two digits, and then four digits, we flag the number as a SSN and declare it sensitive. If the user enters a 15-16 digit number (after removing any spaces), we supply the number to the Luhn algorithm [4], a scheme that serves as a checksum on credit card numbers. If the algorithm indicates the number could be a valid credit card number, we flag the data as sensitive. If either heuristic flags sensitive data, we issue a warning if TLS is not in use. These two examples illustrate how browser extensions can leverage both the data requested by forms as well as the data supplied by the user to develop effective heuristics for detecting sensitive data.

In order to evaluate the effectiveness of our implementation,

[1]This extension is available at `http://www.cs.indiana.edu/cgi-pub/cshue/research/formcheck.php`.

we manually examined Web pages not contained in our data to characterize whether they contained forms requesting sensitive data and whether our heuristics for identifying them will flag them as such or not. To do so, we randomly selected 43 Web pages from the Alexa service that were not included in our previous data sets for generating the heuristics. Collectively, these Web pages contained 100 forms which we examined manually. From this inspection, we found 22 forms which requested sensitive data, all of which were detected by the heuristics in our implementation. An additional 2 forms requested location information which could be considered sensitive. Neither of these were detected by the heuristic; however, adding the term "location" as a keyword in the heuristic would have caused them to be detected. In no cases did the heuristic indicate data was sensitive when it was not.

Our heuristics for identifying sensitive data are based on the observation that Web sites typically use certain conventions and key words in naming fields in their forms. While our tests show that leveraging those conventions works well in practice, a Web site determined to not be flagged while asking its clients for sensitive data insecurely can defeat them. This can sometimes happen without adversarial intentions as well. For example, our heuristics currently leverage the English language to look for input fields requesting sensitive data; however, other languages may be used by the Web site, causing our heuristic to miss some fields. Other factors used by our heuristics are not so easily foiled. The use of "password" as a `type` for `input` fields is standardized in HTML; replicating the obscuring functionality of a password field is unlikely to happen accidentally. However, adversarial sites could still use rich media to emulate such behavior without being detected.

IV. Incorrect TLS Usage in Forms

The second question we sought to answer was: *Do sites that use TLS do so correctly?* Many Web sites contain forms on their main pages. These pages are sometimes delivered without using TLS to avoid paying the performance overhead of TLS connection establishment in cases where users do not log on. Instead, they only use TLS for the actual submission of the form, a practice called *secure post*. Unfortunately, by not authenticating the original form page, the contents of the page can be altered by an attacker before it is delivered to the user. Thus, an attacker can change the form's destination or embed client-side scripts that will leak sensitive data. This can lead to identity theft, with significant costs for both consumers and institutions. Browsers typically indicate when a page is secure by showing a "lock" icon or using color highlighting. However, under secure post, neither of these indicators are present, since the form page is not transmitted under TLS.

Any time a page is delivered through HTTP and contains a form transmitted through HTTPS, secure post is a concern. Our data contained many such instances. *Of insecure pages that had forms, 1.65%-4.49% had at least one form that was submitted via HTTPS.* However, secure post is only problematic when sensitive data is being transmitted through the form. Accordingly, we examine how often secure post is an issue when sensitive data is being transmitted. We use the same heuristics to identify sensitive data as in Section III. In Table III, we show the various types of data requested in forms with the secure post problem. Passwords dominated, as they did with insecure forms. However, unlike insecure forms, where user name was the third most popular field, user name was the second most popular field in secure post forms in most of the data sets. Account number and credit card number were still not requested as frequently as the other data types in most data sets. However, in the popular sites, account numbers were requested more than phone numbers or inferred passwords. With other fields, we noticed no considerable differences in the types of data requested.

TABLE III
THE NUMBER OF PAGES WITH DIFFERENT CATEGORIES OF SENSITIVE DATA REQUESTED BY UNAUTHENTICATED HTTPS FORMS

Field Type	DMOZ Breadth	DMOZ Depth	Alexa	DNS
Password Fields	8,871	419	4,666	7,927
Inferred User Name	4,185	195	2,100	3,699
Inferred Address	2,826	39	1,878	1,096
Inferred Email	2,825	104	1,590	3,881
Inferred Real Name	2,768	46	859	2,321
Inferred City	1,644	18	453	774
Inferred Password	154	1	215	116
Inferred Phone Number	123	10	50	301
Inferred Account	76	2	297	18
Inferred Card Number	13	4	5	73

A. Browser Extensions to Overcome Incorrect TLS Usage

One approach to avoid the pitfalls of secure post is to verify the original HTTP page containing the HTTPS form before submitting any data through the form. This validation would authenticate the Web server and ensure that the original page was not modified. Once this verification is done, TLS usage in the form would ensure the confidentiality of data.

To validate the original HTTP page via an HTTPS connection, we implemented an extension to the Mozilla Firefox Web browser. In our implementation, before submitting any form data, the client establishes a TLS connection with the server providing the page, obtaining the server's certificate and authenticating the server. The extension then requests the sign-in page through the TLS connection; if the HTML code for the original sign-in page exactly matches the code provided by the unprotected page, the extension submits the completed form to the server. Otherwise, the extension warns the user that the page could not be validated and explains the risks involved. This approach has the advantage of not requiring modifications on the Web server and is a fairly light-weight implementation on the client. To evaluate the approach, we examined 32,230 sites that used secure post in our Web crawling. We obtained each page using an unprotected HTTP connection and then attempted to download the same page using a TLS-protected HTTPS connection. About 71% of Web servers did not have an HTTPS server running, causing the approach to fail. Of the remaining 29% of sites, the pages matched 8% of the time and did not match 92% of the time. From a manual inspection of a subset of the sites that do not match, many seem to be blank pages or indicate the server is not configured to display a Web site. These results show that while a fully client-side solution may help in some cases, it currently does not seem to be effective for the vast majority of sites.

An alternate approach we propose is for Web servers to hash and sign their login page in advance and provide its signature while serving the page. Before submitting forms on the page, the client would establish a TLS connection with the Web server, obtain the server's certificate, verify the certificate to authenticate the server, and then use the server to authenticate the Web page's signature. If the signature is successfully verified, the client will then submit the form contents supplied by the user to the server. Otherwise, the user would be warned that the signature did not match and the submission would be canceled. The hash and sign approach would be sufficient for static Web servers; however, each time the server changes the content on its main page, a new signature would need to be generated. To avoid this, the signature could be designed to cover the form on the Web page as well as any client side scripts that would interact with the form. The rest of the page could be altered without requiring a new signature. This approach would require support from both the Web servers and clients.

The proposed approaches provide the Web servers with a means to avoid unnecessary TLS transactions and the associated performance overheads while still providing end-users with authenticity when obtaining forms. However, neither approach is a complete solution: if an attacker simply changes the destination of the form to a non-TLS server and strips any Web server signatures (for the second approach), the client may not be able to detect that the page should be further authenticated. Additionally, an attacker could bypass the sensitive data checking from Section III-A by obfuscating the page to avoid detection. To close this attack vector, a third-party could create and sign a list of sites known to employ secure post. This list could be periodically obtained by the Web browser and an alert raised if a page classified as using secure post instead begins submitting pages without any protection. While this approach would not cover all sites, it could be used to decrease the risk to many Web users.

V. Related Work

The security underlying the SSL and TLS protocols has been scrutinized from a variety of angles. The TLS protocol was originally specified in RFC 4346 [1]. Wagner et al. examined the security of the SSL v3.0 protocol and found a few weaknesses in the protocol, but propose simple corrections to overcome them [5]. Netcraft performed a survey of systems using TLS and found that only about 27% of sites used TLS certificates that would not cause a warning [6]. A large portion of the other certificates were either self-signed or did not match the host providing the certificate. Extended validation has been introduced to provide a higher standard of trust that the certificate holder is a legally established and verified organization [7]. Unfortunately, in a study by Jackson et al. [8], extended validation did not decrease the susceptibility of subjects to different types of attacks. In the work by Schechter et al. [9], the authors examine the response by users to missing security indicators and the presence of warning messages. In this study, the authors found that few subjects (4%) did not submit their credentials after some security indicators were missing (no lock icon, missing "https" in the URL, and removal of a site-authentication image). However, when a warning message was present, 44% of subjects did not enter their credentials. With more accurate and context-sensitive warnings, we suspect that fewer participants will submit sensitive data. However, a user study on the topic would be warranted to determine its effectiveness.

VI. Conclusion

In this work, we examined the usage of the TLS protocol on 4.3 million Web pages and found that more than a third either do not use TLS when they should or use it incorrectly. The first is the most common case. We developed client-side solutions that can help the users when Web sites shirk their duties. While these heuristics are imperfect, they at least can warn users of the risks from these insecure Web sites.

Ultimately, the Web site operators are responsible for requesting sensitive data in a secure way. Some sites shirk this responsibility, often unintentionally. In these cases, automated Web crawling, like the kind we perform, can be used to alert Web site operators about their insecure practices. Others may continue to disregard their responsibility due to the performance and administrative overheads associated with secure Web practices. In the past, organizations have used "name and shame" campaigns to effect change. For example, the StopBadware project [10] solicits and publishes reports on bad software in order to encourage the software producers to change their practices. A similar model could be used for sites that do not provide proper protections for their users. These organizations may be more likely to employ proper security when the alternative is a decrease in customer trust.

References

[1] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol," IETF RFC 4346, Apr. 2006.

[2] DMOZ, "Open directory project," http://www.dmoz.org/.

[3] Amazon.com, Inc, "Alexa web information service (AWIS)," 2008, http://aws.amazon.com/awis.

[4] H. P. Luhn, "Computer for verifying numbers," U.S. Patent 2,950,048, Aug. 1960.

[5] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 protocol," in *USENIX Workshop on Electronic Commerce*, 1996.

[6] Netcraft, "Secure server survey," 2006, http://news.netcraft.com/SSL-Survey/.

[7] T. C. . Browser Forum, "Guidelines for the issuance and management of extended validation certificates," 2007, http://cabforum.org/EV_Certificate_Guidelines.pdf.

[8] C. Jackson, D. Simon, D. Tan, and A. Barth, "An evaluation of extended validation and picture-in-picture phishing attacks," in *Usable Security (USEC)*, 2007.

[9] S. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The emperor's new security indicators: An evaluation of Website authentication and the effect of role playing on usability studies," in *IEEE Symposium on Security and Privacy*, 2007.

[10] Berkman Center for Internet and Society, "Stopbadware," http://www.stopbadware.org/.