Protecting (even) Naïve Web Users, or: Preventing Spoofing and Establishing Credentials of Web Sites

Amir Herzberg[1] and Ahmad Gbara
Computer Science Department
Bar Ilan University

Abstract

In spite of the use of standard web security measures, swindlers often clone sensitive web sites and/or present false credentials, causing substantial damages to individuals and corporations. We believe that, to large extent, this is due to the difficulty of noticing when a sensitive web page has incorrect location or is simply unprotected. In fact, we show that several of the largest web sites, ask users for passwords in unprotected pages, making them easy targets; apparently, even the designers did not notice the lack of protection.

Several papers presented web spoofing *attacks, but mostly focused on advanced attacks trying to mislead even a careful expert; some also suggested countermeasures, mostly by improved browser user interface. However, we argue that these countermeasures are inappropriate to most non-expert web users; indeed, they are irrelevant to most practical web-spoofing attacks, which focus on naive users. In fact, even expert users could be victim of these practical, simple spoofing attacks, resulting in identity theft or other fraud.*

We present the **trusted credentials area**, *a simple and practical browser UI enhancement, which allows secure identification of sites and validation of their credentials, thereby preventing web-spoofing, even for naïve users. The trusted credentials area is a fixed part of the browser window, which displays only authenticated credentials, and in particular logos, icons and seals. In fact, we recommend that web sites* always *provide credentials (e.g. logo) securely, and present them in the trusted credentials area; this will help users to notice the* **absence** *of secure logo in spoofed sites. This follows the established principle of* ***branding.*** Logos and credentials may be certified by trusted Certificate Authorities, or by peers using PGP-like `web of trust`.

Existing web security mechanisms (SSL/TLS) may cause substantial overhead if applied to most web pages, as required for securing credentials (e.g. logo) of each page. We present a simple alternative mechanism to secure web pages and credentials, with acceptable overhead. Finally, we suggest additional anti-spoofing measures for site owners and web users, mainly until deployment of the trusted credentials area.

1 Introduction

The web is the medium for an increasing amount of business and other sensitive transactions, for example for online banking and brokerage. Virtually all browsers and servers deploy the SSL/TLS protocols to address concerns about security. However, the current usage of SSL/TLS by browsers, still allows web spoofing, i.e. misleading users by impersonation or misrepresentation of credentials. Swindler can perform web spoofing by clever attacks, which are likely to mislead even technically savvy and wary users, or by simpler techniques, which would still mislead most laymen and probably even many expert users, when not on their guard. Indeed, there is an alarming increase in the amount of real-life web-spoofing attacks, usually using the simpler techniques. Often, the swindlers lure the user to the spoofed web site by sending her spoofed e-mail messages that link into the spoofed web-sites; this is a *phishing attack*. In a typical phishing attack, spoofed spam e-mail messages are lure the victim into spoofed web sites, e.g. impersonating as financial institutions. The goal of the attackers is often to obtain personal and financial information and abuse it for identity theft. A study by Gartner Research [L04] found that about two million users gave such information to spoofed web sites, and that "Direct losses from identity

[1] Contact author; addresses: herzbea@cs.biu.ca.il and http://amirherzberg.com.

theft fraud against phishing attack victims — including new-account, checking account and credit card account fraud — cost U.S. banks and credit card issuers about $1.2 billion last year.”. For examples of phishing e-mail messages, see [Citi04]. Spoofing attacks, mostly using the phishing technique, are significant threats to secure e-commerce, see e.g. [APWG04, BBC03] and Figure 1. We investigate spoofing and swindling attacks and present countermeasures, focusing on solutions that protect naïve as well as expert users.

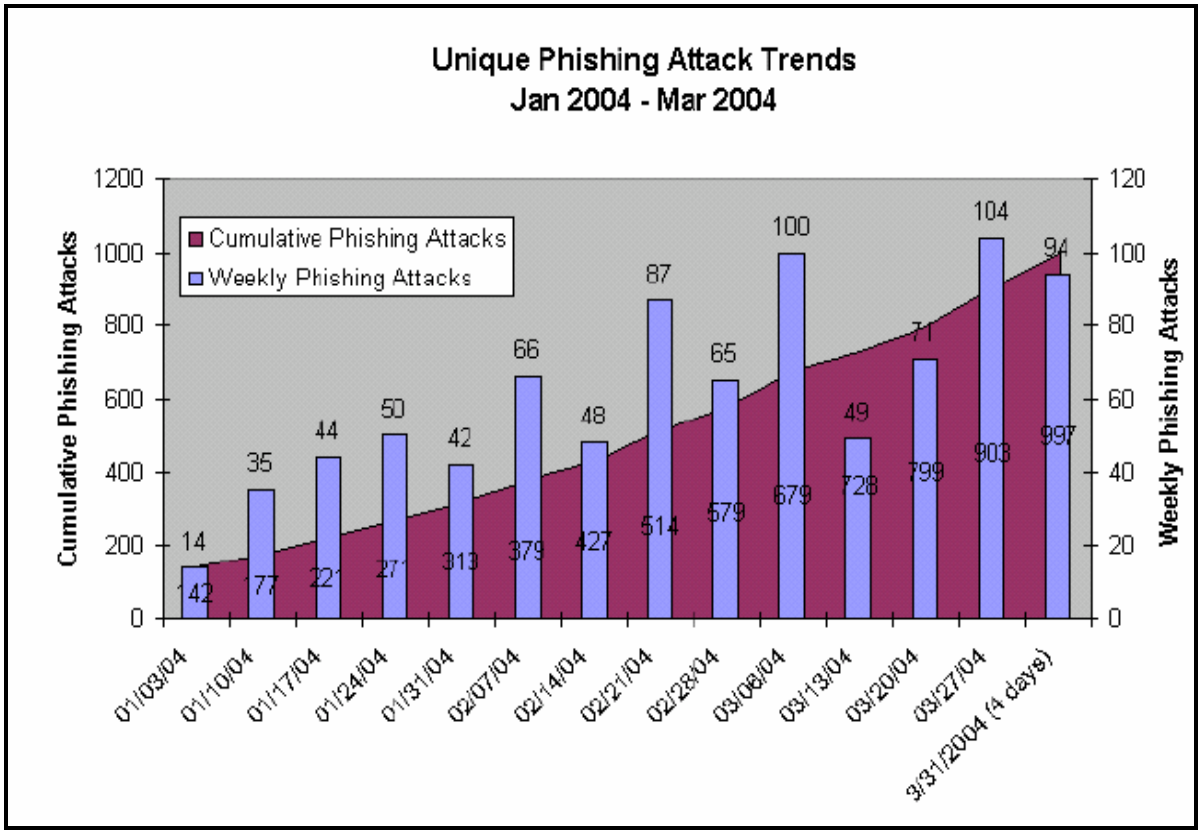


Figure 1: Phishing attack trends (source: [APWG04])

Felten et al. first identified the web-spoofing threat in [FB*97]. However, in this work and all follow-up works [LN02, LY03, SF03, YS02, YYS02], the focus was on attacks and countermeasures for knowledgeable and wary users, who check indications such as the URL (location) of the web site and the security lock (SSL/TLS) indicator. However, practical web-spoofing attacks deployed so far, do not use such techniques at all, or use just basic scripts to present fake location bar [APWG04, Citi04]. Indeed, we argue that most users will not be able to detect well-designed spoofed web sites, even without requiring the attacker to emulate browser functionality at all. Such attacks will therefore succeed even using the countermeasures proposed in the existing literature [LN02, LY03, SF03, YS02, YYS02].

To prevent web spoofing, we propose to establish a *trusted credentials area* of the browser window, in which the browser displays validated logos, seals and other credentials of the web page / site. We recommend that commercial and organizational web sites present secure logo and credentials in *all* of their web pages, both in order to protect the integrity of these pages, and to increase the likelihood of users detecting a spoofed (sensitive) web page, by noticing the *lack* of the appropriate logo and/or credentials in the trusted credentials area.

We use cryptographic mechanisms to validate the logos and credentials in the trusted credentials area. Specifically, we show how to use the (existing, deployed) SSL/TLS protocols for this purpose. However, currently many organizations use SSL/TLS only for few sensitive web pages, since these protocols may involve

substantial overhead if applied for most or all web pages. To allow web-sites to protect the integrity of *all* of their pages, as we recommend, we present an alternative mechanism, the connection-less transport layer security (CLTLS) protocol. CLTLS protocol is a simple variant of TLS, but by being connection-less and by additional optimizations, it can provide the necessary validation of web pages with acceptable overhead.

Our solutions are simple and practical; we are implementing them as extensions of the Mozilla open-source browser [Mozilla]. We hope that this publication will allow additional implementations as well as feedback that will help us improve the design.

1.1 Organization

We begin, in Section 2, with a review of web spoofing attacks and defenses, deriving a set of design criteria for protecting naïve users against spoofing (and phishing) attacks. Then, in Section 3, we present the trusted credentials area approach. In Section 4 we present the connection-less transport layer security (CLTLS) protocol, required to ensure acceptable overhead of validation of credentials and logos. Finally, in Section 5, we conclude with discussion of future work and recommendations for web-site owners, end users, and browser developers.

2 Web Spoofing: Threat models, Attacks, Defenses and Design Criteria

The initial design of Internet and Web protocols assumes benign environment, where servers, clients and routers cooperate and follow the standard protocols, except for unintentional errors. However, as the amount and sensitivity of usage increased, concerns about security, fraud and attacks became important. In particular, since currently Internet access is widely (and often freely) available, it is very easy for attackers to obtain many client and even host connections and addresses, and use them to launch different attacks on the network itself (routers and network services such as DNS) and on other hosts and clients. In particular, with the proliferation of commercial domain name registrars allowing automated, low-cost registration in most top level domains, it is currently very easy for attackers to acquire essentially any unallocated domain name, and place there malicious hosts and clients. We call this the *unallocated domain adversary*: an adversary who is able to issue and receive messages using many addresses in any domain name, excluding a finite list of (already allocated) domain names. While we are not aware of prior definition of this weak form of adversary, we believe most experts would agree that this is the most basic and common type of adversary, and that Internet applications and mechanisms should be resilient at least to attacks by unallocated domain adversaries.

Unfortunately, we believe, as explained below, that currently, most (naïve) web users are vulnerable even against unallocated domain adversaries. This claim may be surprising, as sensitive web sites are usually protected using the SSL or TLS protocols, which, as we show in the following subsection, securely authenticate web pages even in the presence of *intercepting adversaries,* which are able to send and intercept (receive) messages from *all* domains. Indeed, even without SSL/TLS, the HTTP protocol securely authenticates web pages against *spoofing adversaries,* which are able to send messages from all domains, but receive only messages sent to unallocated (adversary-controlled) domains. However, the security by SSL/TLS (against intercepting adversary; or by HTTP against spoofing adversary) is only to the requesting application (usually browser), and only with respect to the specific address (URL) and security mechanism (HTTPS, using SSL/TLS, or `plain` HTTP); what guarantees that this address and security mechanism really correspond to the user's intentions and expectations? Web spoofing attacks usually focus on this gap between the intentions and expectations of the (naïve) user, and the address and security mechanism specified by the browser to the transport layer.

In the next subsection, we give a very brief description of the SSL/TLS protocols, focusing on their mechanisms for server authentication. We then review Web-spoofing and phishing attacks, showing how they are able to spoof even sensitive web sites protected by SSL/TLS. We also discuss some of the countermeasures against web spoofing proposed in previous works, and argue that they are appropriate for security savvy and alert

users, but may not be sufficient for naïve, off-guard users. We complete this section by identifying design criteria for defenses against web spoofing.

2.1 Server Authentication with SSL/TLS

Netscape Inc. developed the Secure Socket Layer (SSL) protocol, mainly to protect sensitive traffic, such as credit card numbers, sent by a consumer to web servers (e.g. merchant sites). Transport Layer Security (TLS) is the name of an IETF standard designed to provide SSL's functionality; most browsers enable by default both SSL and TLS. TLS has several improvements in cryptographic design, but they are beyond the scope of this article (see [R00, H04]); therefore we use, from here on, the name SSL, but refer also to TLS.

We focus on SSL's core functionality and basic operations. Simplifying a bit, SSL operation is divided into two phases: a handshake phase and a data transfer phase. We illustrate this in Figure 2, for connection between a client and an imaginary bank site (*http://www.bank.com*). During the handshake phase, the browser confirms that the server has a domain name certificate, signed by a trusted Certificate Authority (*CA*), authorizing it to use the domain name *www.bank.com* contained in the specified web address (URL). The certificate is signed by *CA*; this proves to the browser that *CA* believes that the owner of the domain name *www.bank.com* is also the owner of the public key *PKserver*. Next, the browser chooses a random key *k*, and sends to the server $Encryt_{PKserver}(k)$, i.e. the key *k* encrypted using the public key *PKserver*. The brower also sends $MAC_k(messages)$, i.e. Message Authentication Code using key *k* computed over the previous messages. This proves to the server that an adversary didn't tamper with the messages to and from the client. The server returns $MAC_k(messages)$ (with the last message from the browser added to *messages*); this proves to the browser that the server was able to decrypt $Encryt_{PKserver}(k)$, and therefore owns *PKserver* (i.e. it has the corresponding public key). This concludes the handshake phase.

The data transfer phase uses the established shared secret key to authenticate and then encrypt requests and responses. Again simplifying, the browser computes $Encrypt_k(Request, MAC_k(Request))$ for each *Request*, and the server computes $Encrypt_K(Response, MAC_K(Response))$ for each *Response*. This protects the confidentiality and integrity of requests and responses.

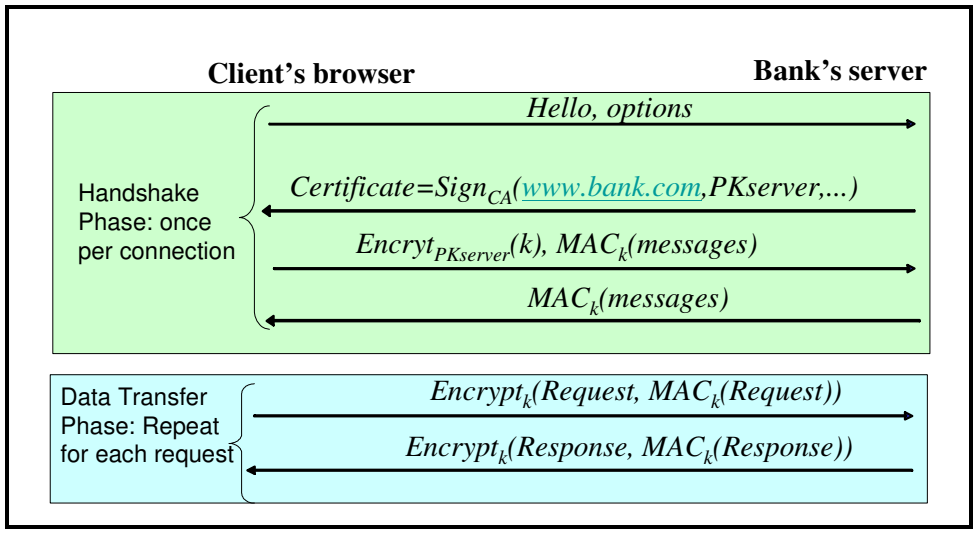


Figure 2: Simplified SSL/TLS Operation

To summarize, web security is based on the following basic security services of SSL:

1. Server (domain name) authentication[2]: confirming that the server has the private key which can decrypt messages encrypted by the client using public key *PKserver*, where the certificate signed by the CA establishes the linkage between the ownership of *PKserver* to the domain name of the site (www.bank.com in this example).
2. Confidentiality and authentication of the traffic between client and server, by using encryption and message authentication (MAC) using the shared secret `master key` established during handshake phase.

Unfortunately, most web pages are *not* protected by SSL. This includes most corporate and government web pages, and other sensitive web pages. The reason is mainly performance; the SSL protocol, while fairly optimized, still consumes substantial resources at both server and client, including at least four flows at the beginning of every connection, state in the server, and computationally-intensive public key cryptographic operations at the beginning of many connections. Later on, we present a more efficient mechanism for authenticating credentials of web pages and other objects.

2.2 Web-spoofing and Phishing Attacks

SSL is a mature cryptographic protocol; while few weaknesses were found in some early versions and implementations of it, the versions currently used, from version 3.0 of SSL and 1.0 of TLS, seem secure. (This refers to the full protocol; the simplified description above is not secure.) However, the security of a solution based on SSL/TLS depends on *how* the protocol is used, e.g. by browsers. There are two major vulnerabilities in the way browsers use SSL/TLS.

The first vulnerability is due to the dependency on public key certificates linking the public key with the location (URL). In the *false certificate attack*, the adversary receives a certificate for the domain of the victim web page from a CA trusted by the browser, but containing a public key generated by the adversary. Therefore, the adversary has the matching private key and can pass SSL server authentication for the victim web page. Most browsers are pre-installed with a long list of certification authorities which are trusted for server authentication by default; few users inspect this list and remove unknown or untrusted CA entries. Furthermore, since CA compete with each other on offering certificates to servers, and have very limited if any liability in case of fraud, it is usually fairly easy and inexpensive to obtain false site (and e-mail) certificates; see [ES00, FS*01]. We are not aware of swindlers actually deploying this attack in practice so far, possibly due to the existence of the larger vulnerability that we next describe.

The second, larger vulnerability is that due to the dependency on the user to validate the authenticity of web sites, by noting relevant status areas in the browser user interface. The relevant status areas are the location bar, containing the URL (Universal Resource Locator), and the SSL/TLS indicator (typically, as open lock for insecure sites, closed lock for SSL/TLS protected sites). We are mostly interested in the *web spoofing attack,* which exploits this vulnerability, by directing the browser to an adversary-controlled *clone site* that resembles the original, *victim site*, which the user wanted to access. Web spoofing attacks are very common, and are the most severe threat to secure e-commerce currently. As we explain below, most web spoofing attackers simply rely on the fact that most users would usually not notice a wrong URL or SSL indicator, when approaching their online banking site (or other sensitive site). Therefore, an attacker can circumvent the SSL site authentication trivially, by not using SSL or by using a URL for which the attacker can obtain a certificate. More advanced attacks can mislead even users that validate the SSL indicator and location bar (containing URL).

The first challenge for a web spoofing attack is to cause the browser to receive the clone site, when the customer is really looking for the victim site. The attacker can exploit different parts of the process of receiving a (sensitive) web page. We illustrate the typical scenario of receiving a sensitive web page in Figure 3. The process begins when the user selects the web site, by entering its location (URL) or by invoking a bookmark or link, e.g. in an e-mail message (step 1a). The browser, or the underlying transport layer, then sends the name of the domain

[2] SSL also supports client authentication, but this is rarely used to protect web transactions, possibly due to concerns about user acceptance and support costs; we therefore do not discuss it.

of the site, e.g. xxx.com, to a Domain Name Server (step 2a). The Domain Name Server returns the IP address of the site (step 2b). Now, the client sends an HTTP request to the site, using the IP address of the site (step 3a), and receives the HTTP response containing the web page (step 3b); these two steps are protected by SSL, provided that the URL indicates the use of SSL (by using the *https* protocol in the URL). Finally, the browser presents the page to the user (step 1b).

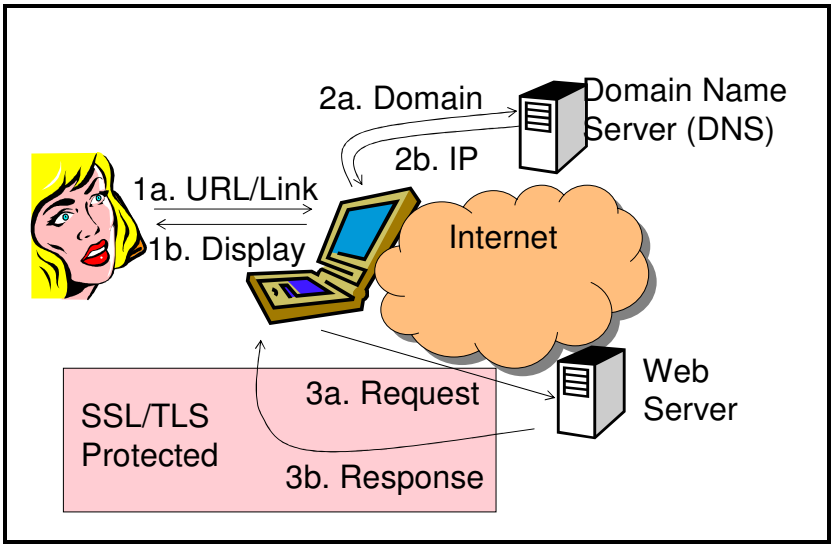


Figure 3: HTTP request/response process with SSL protection

If we did *not* use SSL/TLS, the attacker could attack all three pairs of steps in this process, as follows:

1. Trick the user into requesting the spoofed web site in step 1a, and/or into using *http* rather than *https*, i.e. not protect the request and response using SSL/TLS.

2. Return an incorrect IP address for the web server in step 2b. This can be done by exploiting one of the known weaknesses of the DNS protocol and/or of (many) DNS servers. A typical example is DNS cache poisoning (`pushing` false domain→IP mappings to the cache of DNS servers).

3. Intercept (capture) the request in step 3a (sent to the right IP address) and return a response in step 3b from the spoofed site.

The goal of the server authentication mechanism in SSL/TLS, briefly outlined above, is to prevent the two last attacks, by checking that the server's certificate contains the requested domain, as long as the certificates are valid. Indeed, browsers using SSL and/or TLS will detect, and warn the user, if the site does not have the private key matching the public key in a valid certificate signed by a certificate authority (CA) whose public key is in the list kept by the browser. An attacker may use a false certificate from one of the many certificate authorities in the default list of popular browsers (e.g. 115 in Internet Explorer 6.0). Furthermore, since different browsers maintain different default lists of certificate authorities, and certificates are expensive, users often receive warnings about certificates (expired, signed by unknown CA key, etc.). As a result, it is quite likely that users will ignore the warning regarding the certificate; Grigg [G04] reports a spoofing web site, which simply used a certificate from a non-existent CA, relying on users tendency to `click through` warning windows.

However, most spoofing attacks against SSL/TLS protected web sites focus on the first attack, i.e. tricking the user into requesting the spoofed web site and/or into using an insecure connection (without SSL) rather than an SSL-protected connection. Unfortunately, spoofers often succeed in tricking users into using the wrong URL, or not using SSL (i.e. *http* rather than *https*). The reason is simple: users rarely type manually the

address of the sensitive web page. Instead, in most cases, users *link* into the sensitive web page from a *referring web page*, which is usually insecure, e.g. a homepage of the service provider or results from search engine. Very few service providers and search engines use SSL to protect the links. Therefore, an attacker that can intercept the requests in step 3a, or return incorrect IP addresses in step 2b, is usually able to trick the user into requesting the URL of the spoofed site.

However, most web-spoofing attacks exploit easier methods, which do not require either interception of messages to `honest` web sites, or corruption of the DNS response. One method is *URL redirection*, due Felten et al. [FB*97]. This attack begins when the user accesses any `malicious` web site controlled by the attacker, e.g. containing some content; this is the parallel of a Trojan software, except that users use less cautious about approaching untrusted web sites, as browsers are supposed to remain secure. The attack works if the user continues surfing by following different links from this malicious site. The site provides modified versions of the requested pages, where all links invoke the malicious site, which redirects the queries to their intended target. This allows the malicious site to continue inspecting and modifying requests and responses without the user noticing, as long as the user follows links. However, this still requires the spoofer to intercept communication to a web site used by the user, or to control such a site.

In practice, attackers usually use an even easier method to direct the user to the cloned site: *phishing attacks,* usually using spam e-mail messages. In Figure 4 we describe the process of typical phishing attacks. The adversary first buys some unallocated domain name, often related to the name of the target, victim web site. Then, the adversary sends spam (unsolicited e-mail) to many users; this spam contains a `phishing bait message`, luring the user to follow a link embedded in the bait message, e.g. claiming it is a link to an important service from a trusted entity. The link actually connects the users to the spoofed web site, emulating the site of the victim entity, where the user provides information useful to the attacker, such as credit card number, name, e-mail addresses, and other information. The attacker stores the information in some `stolen information` database; among other usages, he also uses the credit card number to purchase additional domains, and the e-mail addresses and name to create more convincing spam messages (e.g. to friends of this user).

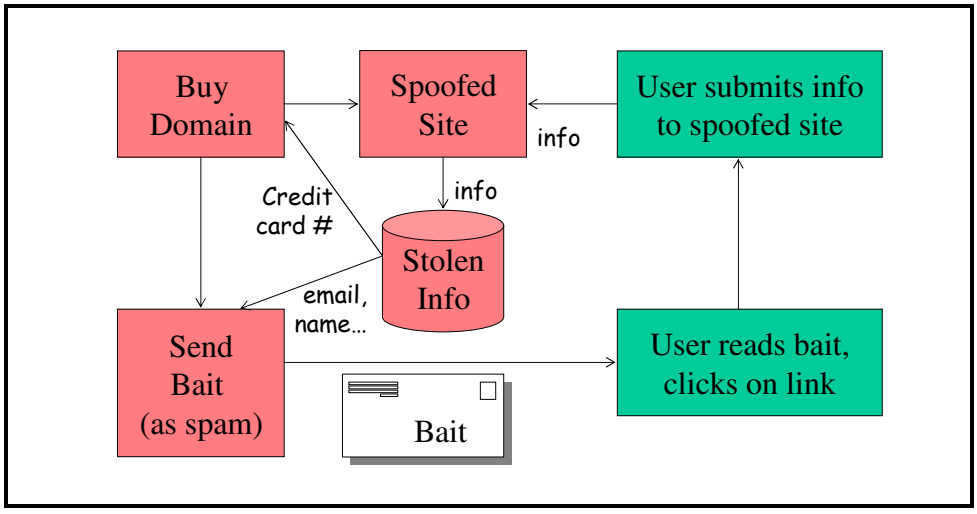


Figure 4: Phishing Attack

Currently most phishing attacks lure the users by using spam (unsolicited, undesirable e-mail), as described above. However, we define *phishing attack* as (any method of) luring the user into directing his browser to approach a spoofed web site. For example, an attacker could use banner-ads or other ads to lure users to the spoofed site. We believe spam is the main phishing tool simply since currently spam is extremely cheap and hard to trace back to the attacker. Spamming is causing many other damages, in particular waste of human time and attention, and of computer resources. Currently, the most common protection against spam appears to be content

based filtering; however, since phishing attacks emulate valid e-mail from (financial) service providers, we expect it to pass content-based filtering. Some other proposals for controlling and preventing spam may also help to prevent or at least reduce spam-based phishing, see e.g. [CSRI04, He04].

Most phishing attacks require only a web address and server, but do not require intercepting (HTTP) requests of the user; therefore, even weak attackers can deploy them. This may explain their popularity, as shown in Figure 1. This means that the domain name used in the phishing attack is different from the domain name of the victim organization.

Many phishing attacks use deceptive domain names similar to the victim domain names, possibly motivating the change by some explanation in the e-mail message, e.g. see [BBC03]; often, they also display the anchor text with the correct victim URL, while linking to the spoofed web site. Unfortunately, most naïve users do not notice the attack or the change in the domain name, since:

- Users do not read the location bar at every transaction.
- Users do not always understand the structure of domain names (e.g., why accounts.citibank.com belongs to CitiBank™, while citibank-verify.4t.com was a spoofed site [Citi04]).
- Organizations often use multiple domain names, often not incorporating their corporate name, e.g. CitiBank™ uses at least eight domains, e.g. accountonline.com.

Since the spoofed web sites use deceptive domain names, they usually could purchase a certificate from one of the (many) Certificate Authorities programmed in the default list of popular browsers; for example, why would a CA refuse to provide a certificate for the 4t domain? However, most spoofed web sites do not even bother to get a certificate (with the costs and overhead involved). Instead, most spoofed web sites – and in particular all of the (many) examples in [Citi04] – simply do not invoke SSL/TLS. We observe that many, or probably even most, users did *not* notice the lack of use of SSL/TLS in these spoofed sites.

In fact, it turns out that many existing web sites require sensitive information such as user ID and passwords, in *unprotected web pages*. This includes some of the most important, well-known and sensitive web sites, including Chase™, Amazon™, Microsoft's .Net Passport and eBay, shown in Figure 5, and others[3], e.g. Yahoo!™ and TD Waterhouse™. For readability, the figure contains only the most relevant part of the web pages, simply by reducing the size of the browser window; the reader can probably find other examples. Examples (a) (Chase™) and (b) (Amazon™) use the Mozilla browser, with our extension, described in the following section, which added a clear warning (on top) noting the fact that these pages are *not* protected by SSL/TLS. This warning would not appear in standard browsers (without our extension), and in fact we noticed the lack of protection in these web sites only after using the browsers with the anti-spoofing extension. In examples (c) (Microsoft .Net Passport) and (d) (eBay™) we used Internet Explorer (without extension); the fact that these are insecure pages is indicated here only by the *lack of the `lock`* icon representing secure sites. All of these pages *encourage* users to input passwords and account numbers, *implying (incorrectly) that these sites are secure*; but an attacker could present a spoofed web-page, which would appear the same and collect this sensitive information. We believe that this proves that users – and even serious web-site designers – do not notice the lack of SSL/TLS protection, even in sensitive web pages belonging to established organizations.

[3] Interested readers can also check directly if these sites are still insecure, as when we visited them on July 2004, by following the following links to Chase™, Amazon™, TD Waterhouse™, eBay™, Microsoft's .Net Passport and Yahoo!™. We hope, of course, that these sites will fix soon this bug (we informed all of them). Readers are encouraged to find additional unprotected sites and inform the owners and us.

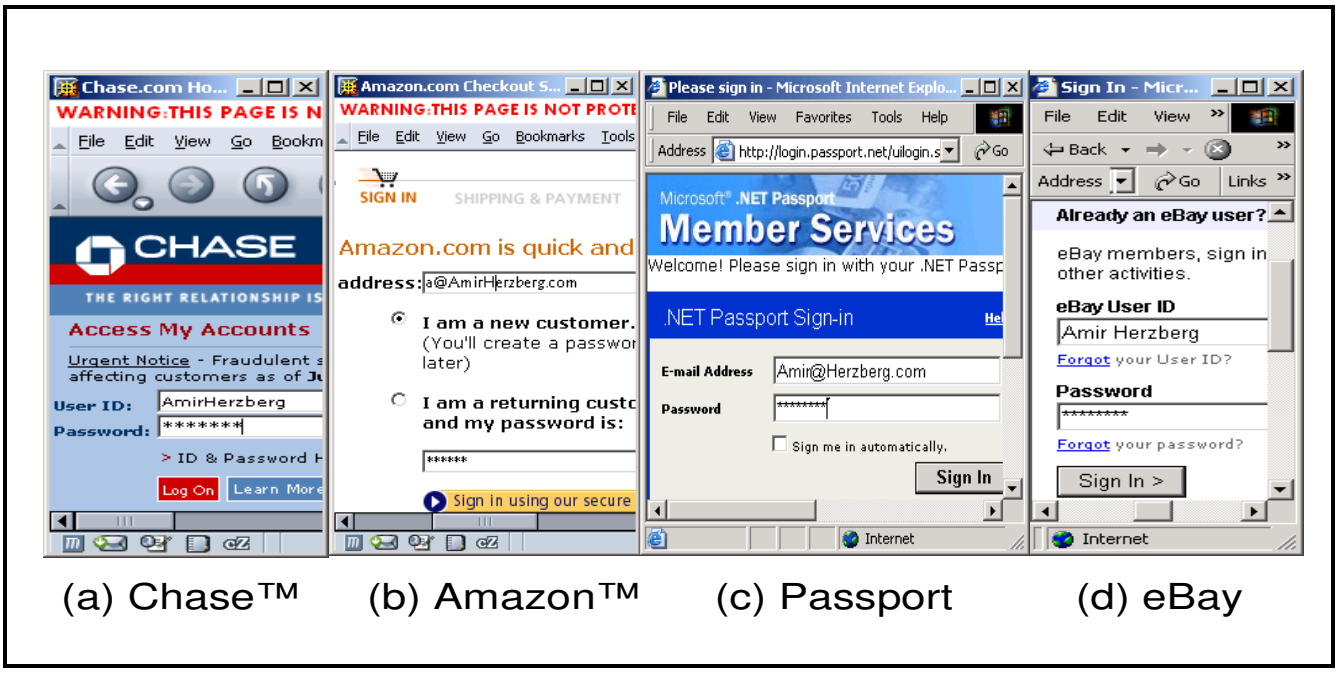


Figure 5: Unprotected Login to Important Sites

In more advanced Web Spoofing attacks, first presented in [FB*97], the adversary uses browser features to make it appear as if the browser displays the SSL-protected victim web page, while in fact it is displaying a cloned page. Some of these attacks are very simple, yet effective. For example, in one deployed attack [Citi04], the attacker opens two browser windows: a small one, which clones Citibank™ login screen and contains no status information or bars, inside a larger one, which is simply the regular Citibank™ web site. Such sites can be extremely convincing; we believe most users will not realize that they enter their password into a separate, insecure window, whose URL is not even displayed. In another deployed attack [APWG04, SF9182], a bug in many Internet Explorer™ browsers is exploited, allowing the attacker to cause false information to be displayed in the location bar.

Several works presented even more elaborate web spoofing attacks, using scripts and/or Java applets, to make it even harder for users to detect the cloning [LN02, LY03, SF03, YS02, YYS02]. These works also propose solutions, by disabling (important) browser functionalities, or using enhancements to the browser UI to make it hard or impossible for the attacker to display spoofed versions of important browser indicators [YS02, YYS02, LN02]. However, as noted by [YS02], these proposals rely on users' understanding their `signals`, which are (even) more elaborate than the existing location bar and SSL indicator. We therefore believe that these proposals are not appropriate for most users, and definitely inappropriate for naïve users. In fact, as we argued above, we believe that the current indications of location and protection (SSL) are not sufficiently visible and significant to most users. Our design strives for much more visible and easy to understand indications, allowing most users to detect when they reach a spoofed web site.

2.3 Credentials Spoofing Attacks

So far, our discussion, as well as most prior research and reported incidents, focused on web spoofing, i.e. spoofing of the identity of the entity owning the web site. We now consider a related threat, which we call *credentials spoofing attacks*. These attacks involve sites that present misleading, unauthorized credentials, e.g. in the form of graphical seals, logos and images. The motivations for credentials spoofing attacks range from obtaining sensitive information, sent only to trusted sites (such as credit card number or personal details which may be used for identity theft), to simply attracting surfers and customers.

To understand the importance of secure credentials for e-commerce, consider open marketplace sites, such as eBay™. Such sites facilitate commerce between consumers and many small businesses; each transaction requires considerable trust by the consumer, who usually pays, in a non-reversible way, before receiving the goods. To establish this trust, eBay™ and other services, e.g. Square Trade™, maintain secure records of the feedback each seller received from its past buyers. There is clearly an incentive for sellers to fabricate and improve their grades, and some are known to do so, e.g. by providing artificial reviews. A similar, well-known situation exists in review sites, e.g. for hotels or for books. In particular, [H04] reports that the (anonymous) reviewers of books in the Amazon™ site are often the authors of the books or of competing books, or other interested parties.

Presentation of false credentials to attract customers is one of the ancient methods of fraud, and common in the real world as well as on the Web. Currently, the only way for web sites to present credentials is by including an appropriate picture (often referred to as a seal), e.g. e.g. eBay's `power seller` seal, Square Trade's logo, or many others. It is quite trivia for an unauthorized site to copy the seal from an authorized site and present it without authorization. In fact, while many sites present some sort of seal (of quality, privacy, security etc.), site owners, and many users, are aware that these seals are not very secure. We believe that this is the reason that seals and credentials are less common in cyberspace (than in physical business).

2.4 Spoofing Prevention: Design Criteria

We now present design criteria for prevention of web and credential spoofing, extending the criteria presented in [YS02].

- **Provide branding, prevent spoofing.** Obviously, the most basic requirement is that credentials, logos or any other identification information should be secure against spoofing by adversaries. Namely, the adversary should not be able to emulate any such information presented by the anti-spoofing mechanism, when the user is viewing a web site unauthorized to present these credentials, logo, or address. In particular, the mechanism should allow organizations to use and reinforce brand identity, as a major mechanism to identify the organization and establish confidence in it and in the web site.

- **Effectiveness for naïve users:** the credentials should be highly visible, which will ensure that even naïve, off-guard users, will detect the *lack of* necessary credentials when accessing a web site. In particular, as indicated by [YS02], graphical indicators are preferable to textual indicators, and dynamic indicators are preferable to static indicators. Furthermore, to facilitate recognition by naïve users, the credentials should use simple, familiar and consistent presentations. Finally, and again as indicated by [YS02], the (secure) browser should couple between the indicators and the content, rather than present them separately.

- **Support all kinds of credentials:** it should be possible to protect any credential, including information currently displayed in browser status areas (location, SSL indicator, etc.) and additional credentials such as logos, seals, certificates etc.

- **Minimize/avoid user work:** The solution should not require excessive efforts by the user, either to install or to use. In particular, we prefer to base credential validation on simple visual clues, without requiring any conscious user activity during validation. This is both to ensure acceptability of the mechanism, as well as to increase the likelihood of detection of the lack of proper credentials by naïve users.

- **Minimize intrusiveness:** the solution should have minimal or no impact on the creation of web sites and presentation of their content.

- **Customization:** the visual representation of the different credentials should be customizable by the user. Such customization may make it easier for users to validate credentials, e.g. by allowing users to use the same graphical element for categories of sites, for example for `my financial institutions`. Similarly, a customized

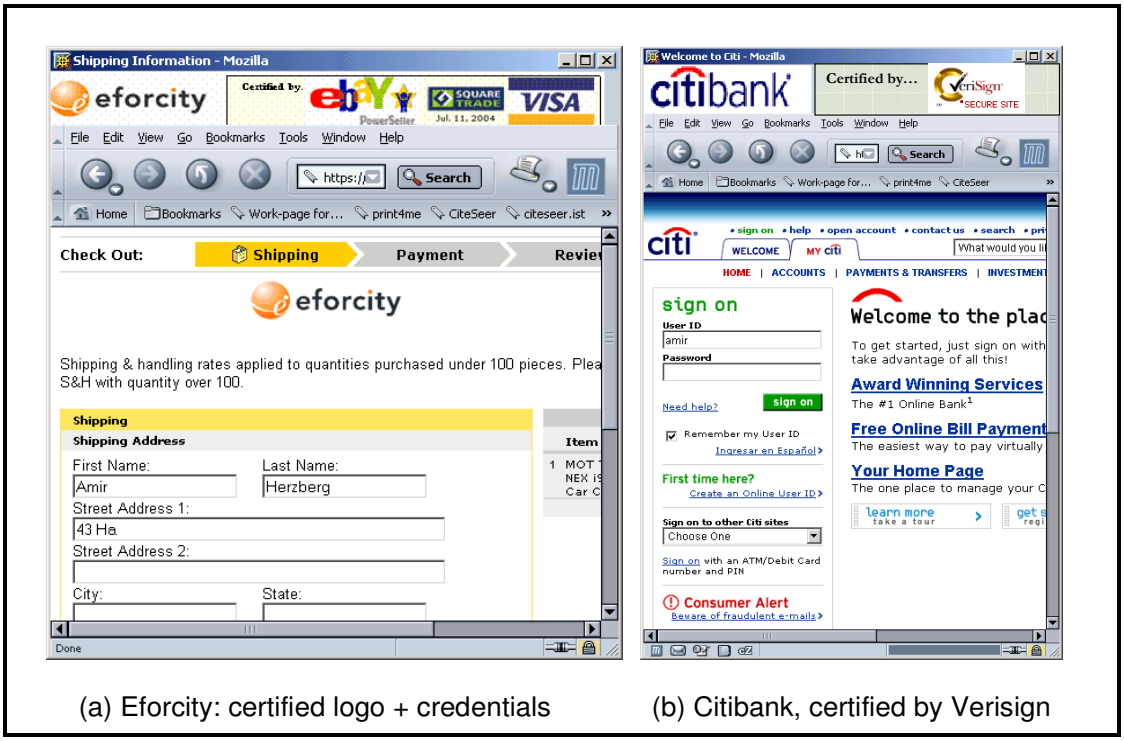
policy could avoid cluttering the trusted credentials area with unnecessary, duplicate or less important logos; e.g., it may be enough to present one or two of the credit card brands used by the user (and that the site is authorized to accept), rather than present the logos for all of them. In addition, customization could allow users to assign easy to recognize graphical elements (`logos`) to sites that do not (yet) provide such graphical identification elements securely (i.e. that do not yet adopt our proposals). Finally, as argued in [YS02], by having customized visual clues, spoofing may become harder.

- **Migration and interoperability:** the solution should provide benefits to early adopting sites and consumers, and allow interoperability with existing (`legacy`) web clients and servers. In particular, it should be sufficient for a client to use the solution, to improve the security of identification of existing SSL/TLS protected web sites.

3 Preventing Spoofing with Trusted Credentials Area

We suggest adding a new component to the user interface of the browser, which we call the *trusted credentials area (TCA).* The goal of the trusted credentials area is to present highly visible, graphical interface, establishing securely the credentials and identity of the web site and of the content presented. We expect that the browser will present most credentials and identifiers via graphical elements such as logos, icons and seals, defined or at least approved by the user or somebody highly trusted by the user (see more below). We implemented the *Trusted Credentials Area (TCA) browser extension* for the open-source Mozilla™ browser; see screen-shots of two protected sites, with logos and other credentials presented in the TCA, in Figure 6. We refer to a browser supporting a TCA, natively or via an appropriate browser extension, as *TCA-enabled.*

Our main design decision was that the trusted credentials area should be a significant area, located at the top of the browser window, and large enough to contain highly visible logos and other graphical icons for credentials. Furthermore, the trusted credentials area must appear in *every* web page, protected or unprotected. In fact, by using the very top area of the window, it is relatively easy to prevent scripts and applets from writing over the trusted credentials area, thereby preventing spoofing. It is also easy to see that this design meets all the other criteria above.



(a) Eforcity: certified logo + credentials (b) Citibank, certified by Verisign

Figure 6: Screen-shots of secure sites with logo in Trusted Credentials Area

The browser should protect the trusted credentials and logos area from spoofing, by preventing scripts and helper windows, including applets, from removing it and displaying fraudulent credentials and logos in (the real or camouflage) trusted area. We achieve this by defining the trusted credentials area in every window opened by the browser, and giving the browser (and code executed by it) access only to the window below the trusted credentials area. This implementation is easy in Mozilla, and seems to be secure against change by any content downloaded from a web server. Additional mechanisms to protect the trusted credentials area include:

1. Helper and applet windows may be marked separately from the browser itself, and in particular from the trusted area, e.g. by enclosing all helper and applet windows by a special, highly visible `warning` border.
2. To make it harder to spoof the trusted area, even for a program that can write on arbitrary locations in the screen, it may be desirable that the background of the trusted area will be a graphical element selected randomly from a large collection, or selected by the user.
3. The browser may restrict opening of new (`pop-up`) windows, including for helper applications and applets. In particular, the browser may confirm that the content was approved by an appropriate authority. This solution can also assist in the prevention of spam web advertisements and pop-up windows.

There are several types of credentials, logos and seals for display in the trusted area; we discuss each of them in the following subsections. Following that, we present the process and architecture for determining the contents of the Trusted Credentials Area.

3.1 Secure vs. Insecure Site Indication

Existing browsers indicate that a site is SSL-protected, by a small SSL-status icon, usually in the status area at the bottom of the page. However, this indication is not very visible, and naïve or off-guard users may not notice its absence, when accessing a sensitive site (e.g. if visiting this site routinely, as for personal bank). Indeed, all of the real-life spoofing (and phishing) attacks we have seen, directly on the Web or described by others (e.g. [Citi04]), were on sites without SSL/TLS protection – even in cases where the attackers used domain names which do not appear related to any known tradename (e.g. 4t.com, used in one of the phishing attacks on

Citibank™). Furthermore, as we show in Figure 5, many trustworthy, important web-sites actually ask users to enter sensitive information such as passwords, in insecure web pages, probably by mistake. All of this shows that the SSL indicator is not sufficiently visible. Furthermore, a web site can request that the browser avoid displaying the status area (simply by using the `window.open` JavaScript method), making the lack of SSL even harder to notice; as mentioned above, swindlers already exploited this method to spoof secure web sites.

To prevent these threats, whenever our browser extension detects that a web site is *not* SSL-protected, it displays a highly visible warning message in the trusted credentials area. We recommend that corporate and other serious web sites avoid this warning message, by protecting *all* of their web pages, preferably presenting the corporate logo in the trusted credentials area. Having all web pages secure could cause a performance problem when security is using SSL or TLS; when this overhead is a problem, one could secure the web pages using the CLTLS protocol presented in next section. By protecting all of their pages, such sites will make it quite likely that their users will quickly notice the warning message in the trusted browser area, when the user receives a spoofed version of a web page of such sites. Furthermore, this ensures that all the organization's web pages will present the logo and credentials of the site (and organization) in the trusted credentials area, using and re-enforcing the brand of the organization.

3.2 Identification of Web Sites

The most basic credential of web sites is the identification of the provider of the web pages and objects. Currently, browsers identify the provider of the web page by indicating the Universal Resource Locator (URL) of the web page in the location bar of the browser. This usually allows (knowledgeable) web users to identify the owner of the site, since the URL includes the domain name (which an authorized domain name registrar allocates to a specific organization). However, the identity of the provider is not necessarily included (fully) in the URL, and the URL contains mostly irrelevant information such as protocol, file, and computer details. Furthermore, the URL is presented textually, which implies that the user must make a conscious decision to validate it. Finally, popular browsers are pre-configured with a list of many certification authorities, and the liabilities of certificate authorities are not well defined; as a result, it may not be very secure to use the URL or identity from the SSL certificate. Therefore, we prefer a more direct and secure means of identifying the provider of the web page, and not simply present the URL from the SSL certificate in the trusted credentials area.

We decided to identify the provider of the web page (or other content) using a graphical symbol, such as logo or icon. The browser extension could include a list of icons for typical sensitive sites, such as `my bank`, `my broker`, `my health-care provider`, and allow the user to select another icon by providing graphical file. However, we believe a logo, selected by the web-site or by the user (e.g. from the graphics in the web page), may be even more natural, effective and popular. Experience and research proved the effectiveness of logos and other trademarks for ensuring recognition (`branding`), including subconscious alert for lack of logo or incorrect logo. In our case, users are likely to notice a missing or incorrect logo when accessing spoofed sites (which do *not* have the correct logo). Logos are also focused on the identity of the provider of the content, rather than on other technical aspects of the URL. Finally, logos are familiar to people, and protected by existing laws and national and international agencies, e.g. the US Patents and Trademarks Office (www.uspto.gov), who also define processes to avoid misleadingly-similar logos and resolve disputes regarding logos.

For SSL and TLS protected web pages, the browser can use the site's public key to identify the logo, since SSL/TLS ensures that the web site has the corresponding private key. Notice this does *not* depend on the identity (domain name or URL) in the certificate; we use alternative mechanisms to link between the public key and the logos or credentials presented. Specifically, we suggest using a public key certificate that will link between the public key of the site and the logo. We call this a *logo certificate,* and we use the term *Logo Certification Authority (LCA)* for an issuer that signs logo certificates.

In our prototype browser extension, the browser extension maintains a *logo certificates folder* where it maintains logo certificates, linking public keys with logos; initially, this folder is empty. In addition, our

extension generates, upon installation, a private signature key, which it uses later on to sign logo certificates, linking public keys and logos, if the user (manually) specifies the use of the logo for the public key. Our browser extension also maintains a list of public keys of trusted Logo Certificate Authorities (LCA), initially containing only its own (self-generated) public key. Finally, for each trusted LCA, the browser extension also maintains its logo and name, to identify it to the user (as responsible for the matching between the site and the site's logo).

After SSL/TLS completes, our browser extension looks in the logo certificates folder for a logo certificate containing the public key of the site (validated by SSL/TLS). If the browser extension finds this public key in one of the logo certificates, properly signed by one of the trusted logo certificate authorities, then it displays the logo of the site, and (optionally) also of the LCA, in the trusted area. For example, Figure 6 shows two SSL-protected sites, where our browser extension presents logos in the Trusted Credentials Area (which, in our implementation, is at the very top of the window). Notice that the same logos are included in the site itself, but this is not secure – an imposter could display the same logos in their site, with or without SSL. Also, while our browser extension maintains the original SSL indicator unchanged, notice how the logos are more visible. Recall that when visiting an insecure site, a TCA-enabled browser fills the entire trusted credentials area (containing the logos in Figure 6) with a very visible warning message/indicator, as shown in Figure 5 (a) and (b).

When the TCA browser extension does not find a logo certificate for the site's SSL public key in the *logo certificates folder*, the site may still indicate the existence of a logo. One way for the site to identify the logo, by location and hash, is using an optional extension of the public key certificate passed to the browser during the SSL handshake phase. Alternatively, and to allow the use of trusted logos using existing SSL certificates, the location (URL) and the hash of the logo may be defined within the page, e.g. using a <META> tag. Finally, we found that often the logo file is easily guessable as one of the images embedded in the page, often containing `logo` as part of the object name. In all cases, the TCA-enabled browser must validate that the site received authorization to display the logo.

One way for sites to prove that they have the necessary credentials to present a logo, is by including with the logo also a special *logo certificate*[4] allowing the display of the logo for sites passing SSL handshake with the given public key. Logo certification authorities issue logo certificates; they should confirm that the owner of the public key in the logo certificate has the legal rights to the logo, and in particular, that logos are not only unique, but also sufficiently distinct to prevent misleading users. As we noted above, national and international trademark registration offices, such as the United States Patent and Trademark Office www.upspto.gov, already provide such services. Such agencies may issues logo certificates directly. Alternatively, such agencies may provide a database of approved logos (and other trademarks), which they confirmed are easily, visibly identifiable, together with identification of the logo / trademark owner. This would allow other (e.g. private) organizations and companies, such as existing certificate authorities, to issue logo certificates.

Similarly to the predefined list of public keys of certificate authorities in current browsers, TCA-enabled browsers may contain a predefined list of (the public keys of) LCA entities, preferably allowing the user to edit this list. We expect such lists to contain a relatively small number of logo certifying authorities, known to many users, e.g. the USPTO. Servers may obtain multiple logo certificates, and logo certifying authorities could cross-certify each other, e.g. when the two issuing organizations cooperate to avoid similar, misleading logos and trademarks.

The user can configure the browser to display logos certified by a LCA automatically, or after prompting the user at the first time. Furthermore, the TCA-enabled browser should also display the logo of the LCA (or multiple logo certifying authorities). For example, in Figure 6 (a), eforcity.com is certified by eBay, Square Trade™ and VISA™, while in (b), Citibank™'s logo is certified by Verisign™. By displaying the logo of the Logo Certifying Authority (eBay, Square Trade, Visa and Verisign), we make use and re-enforce its brand at the same time. Furthermore, this creates an important linkage between the brand of the LCA and the validity of the

[4] Using X.509 certificates and/or terminology, this could be either a public key certificate or an attribute certificate.

site; namely if a LCA failed and issued a certificate for a spoofing web site, the fact that it failed would be very visible and it would face loss of credibility as well as potential legal liability.

To `bootstrap` the process of adoption of the Trusted Credentials Area (TCA) and of logo certificates, we propose that TCA-enabled browsers would also allow *user-certified logos*. Consider a TCA-enabled browser that detects a SSL/TLS protected web page, possibly with a proposed or detected logo, but without an appropriate logo certificate, or with a logo certificate from an unknown LCA (e.g., self-signed – i.e., issued by the site itself). The browser may present a dialog to the user, displaying the logo (if proposed, or if the browser can guess one from the page), and relevant fields from the public key certificate (such as the name of the organization and of the certification authority, and the validity dates), and the following options:

- Certify the proposed or identified logo as corresponding to the site's public key, and display it whenever accessing a page protected by this public key. For this purpose, the TCA browser extension will generate a private signature key, and a matching public validation key, upon installation; it uses the private signature key to sign the logo and the site's public key, creating a *user logo certificate*. The user can also specify whether to *publish* the certificate, for use by the same user (on other machines) or by any user.

- Present an alternate logo, name or icon whenever accessing pages with this public key. The user will specify the alternate logo, name or icon. The user may have a library of icons for different types of sites, used instead of or in addition to the proposed logo. For example, a user may define a `My Banks` logo or icon, and attach it to all of her financial institutions (possibly in addition to a more specific logo of each institution).

- Present the organization name and/or URL whenever accessing a page protected by this public key, together with the logo (if certified) or name of the certification authority (CA). The name of the organization is taken from the SSL certificate, specifically either from the "0=" part of the subject distinguished name, or from the `SubjectAltName' extension. This approach is easy for the users, as it does not require them to choose a logo at all; but now users do not see the logo of the site itself, but only the name, which may make spoofing possible.

The option of *user logo certificates* provides significant security advantage to early users of the TCA browser extension, even when approaching existing, `legacy` web sites (protected by SSL/TLS). The ability of users to certify logos manually is also valuable for web-sites that are concerned about their identification by web users, but are reluctant to purchase an SSL/TLS certificate from one of the major certificate authorities, due to the cost considerations; we believe there are many (especially non-profit) web sites in this category. Such web-sites could use a *self-signed certificates*, signed using the signing key of the organization itself or using some publicly available signature key. Clearly, a spoofer could sign such certificate. However, if the browser properly warns the user when it receives the self-signed certificate for the first time, and the user approves a logo or defines an icon for the certified public key, then the browser can safely use this public key to identify subsequent re-entry to pages of this organization (and automatically display the right logo). Therefore, every web site should have a public key certificate, from a CA or self-signed, and use it to protect, using SSL/TLS or a comparable protocol, all of its web pages that provide or request information, requiring protection from modification or exposure.

Since existing web sites do not contain logo certificates, our prototype provides user logo certificates. This provides us with convenient and secure identification of sensitive sites that we use personally (e.g. bank) or professionally (e.g. program committee site). However, while opportunistic identification is a good short-term measure, it is not perfect; in particular, it still relies on some user awareness, i.e. on users actually validating the details of the organization and certificate authority. While this reduces the burden on the users, since it happens relatively rarely, there is still the risk of users `clicking thru` these security warnings. Unfortunately, many sites use multiple different public/private key pairs (and certificates), usually with no special reason, which increases this risk and inconvenience the users; we became aware of this since the opportunistic identification pops-up whenever encountering an unknown public key, and this often happens several times in some sites (e.g. Citibank™), while not at all in others.

As mentioned above, whenever the user approves (certifies) a logo as belonging to a particular site and public key, the TCA browser extension prompts the user whether to *publish* the newly created logo certificate. Publishing a user logo certificate simply means placing it in some repository, e.g. an FTP or HTTP site. The TCA browser extension also allows users to specify the location of one or more repositories from which it downloads logo certificates (when needed or periodically).

Finally, the TCA browser extension allows the user to input, or approve, logo certificate validation keys, e.g. of the same user on another machine. This allows a user to certify logos only in one machine (e.g. office) and have them appear automatically in other machines (e.g. home or mobile).

However, the user can also input or approve logo certificate validation keys of logo certification autorities, or of *other users he trusts.* This allows users to share the task of validating logos with trusted friends, similar to the PGP web-of-trust [Z95] model, essentially turning these friends into `tiny logo certificate authorities`. We believe this option will facilitate `grass-root` adoption of logo certificates and TCA, which may expedite the deployment of trustworthy, established logo certificate authorities.

3.3 Site credentials and seals

Many web sites display different credentials from third parties, typically using special graphical elements (`seals`) which are trade-marked by the third parties. Examples include reliability rating for online merchants, e.g. eBay's `power seller` or Square Trade's logo, seals for web sites passing different audits, e.g. for security and privacy, certification from trade organization or regulation authorities, and many more. Currently, these credentials are displayed by the web site as part of the page; no technical means prevents a web site from presenting a seal (representing specific credentials) without proper permission from the owner of the seal.

We propose that browser (or a browser extension as in our implementation) will validate the site/page credentials and seals. This allows the user to specify minimal credentials requirements, e.g. to avoid display of unsolicited advertisement pages (spam), as well as other undesirable content (e.g. insecure sites, sites without sufficient privacy protection, or sites containing offensive content). If the content is not rejected, then the credentials are displayed securely in a trusted credentials area of the browser. For example, Figure 6 (a) shows the site of eforcity™, a web merchant that is selling on the eBay™ marketplace, and has an `eBay stores power seller` accreditation, as well as certification from Square Trade™ and from Visa™. We illustrated how the TCA browser extension could display these credentials, if they were provided by appropriate (public key or attribute) logo/credential certificates.

To validate a credential/seal, the browser must receive appropriate credentials for the public key of the site. In particular, the web site may indicate, e.g. in a <META> tag in the web page, the location of a file containing a certificate from the owner of the seal, where the certificate is given to the site's public key. The certificate will identify the credentials of the site, typically using extension fields; the credentials may specify the graphical image (seal) to display, and/or contain a credential-type attribute, allowing the user to select the graphical representation (icon, seal, etc.) for each type of credential.

While currently seals are normally `owned` and issued by a single organization, we envision users also defining (more complex) graphical representations of the attributes in one or more credentials. Some examples of user-defined seals/icons include:

1. Some of the credentials issued to sites may provide one or more attributes of the site, given as values in numeric (or other) range, such as the number of transactions and the fraction of complaints for online merchants, as maintained e.g. by Ebay™. Each user could define few levels and icons for each, e.g. `reliable` and `avoid`.

2. In some cases, users may require multiple certificates, from different issuers, to consider a site as belonging to some class. For example, a user may define an online merchant as `reliable` only if it has a BBBonline® certificate from the Better Business Bureau®, in addition to appropriate ratings from Ebay™.
3. In other cases, the user may want to validate sites in a certain class, against some lists of `bad` sites or companies. We refer to such information as a negative credential. For example, the user may not define an online merchant as `reliable`, if it finds this merchant listed with poor rating at BizRate.com™. In this case, we cannot rely on a certificate from the merchant site; instead, the user must define to the browser that if the site presented certain (positive) credentials, the browser should also search specific `black-list` sites for negative credentials.

3.4 `Last visit` and other historical indicators

Many operating systems, e.g. Unix, display the time and date of the last login as part of the login process. This allows users to detect unauthorized usage of their accounts, by noting usage after their last authorized login. The Trusted Credentials Area (TCA) could provide such credentials, if desired, by maintaining record of previous access to each site (or using each public key). Each user can indicate in the `TCA preferences` what historical indicators to present (or maintain).

Notice, however, that the history is limited to access via this particular browser and computer. This may make the records of entrance to a particular site less valuable than the operating systems indicators; also notice that detection of spoofing using this mechanism requires users to notice wrong historical information. However, this mechanism may be useful to create differentiation between SSL and logo certificates by different certificate authorities. Recall that common browsers contain a pre-defined list of above a hundred certification authorities and keys; clearly, some of them are major, and widely used, and others may be minor, rarely used, and possibly less trusted. The browser could present some statistics on the number of certificates previously received from the CA of the current certificate; this could help make the user more caution with an unusual certificate authority.

3.5 Process and architecture for determining the contents of the Trusted Credentials Area

We present the process for collecting (positive and negative) credentials in Figure 7; the process is based on the designs of [HM*00, HM04]. The process begins when the browser receives an SSL protected web page. Using available interfaces in Mozilla, we detect this and invoke the *Certificates collector module.*

The certificates collector receives from the browser three inputs: the public key used to validate the page (PK), a URL pointing at additional certificates for this page and public key, provided in the page using the <META> tag, and the certificate provided by the server during the SSL handshake. In addition, the certificate collector uses two configuration files defined in advance by the user (or a software agent trusted by the user). The first file contains a list of certificate collection sites, which the certificate collector must consult for any SSL-protected web page; this `*mandatory collections*` list ensures, in particular, that the user will receive indication of any `negative credentials` for the site – we cannot trust the site to provide negative credentials. The second configuration file used by the certificate collector lists the *trust anchors* of the user, i.e. the public keys of root certification authorities trusted by the user, with indication, for each trust anchor, of what kind of certificates it is trusted to provide.

The certificates collector outputs a list of certificates of the page, usually all using the public key of the SSL certificate of the page (but it may include also other relevant certificates, e.g. of the issuer of a certificate to the public key of the page). It passes this list of certificates to the *Attributes extractor module*. This module extracts from the list of the certificates the attributes of each certificates, as a list of triplets, each containing an issuer public key (*IPK*), a subject public key (*SPK*) and one or more attributes (*Attrs*). Internally, this module performs the conversion of certificates and extraction of attributes in two stages, following [HM*04].

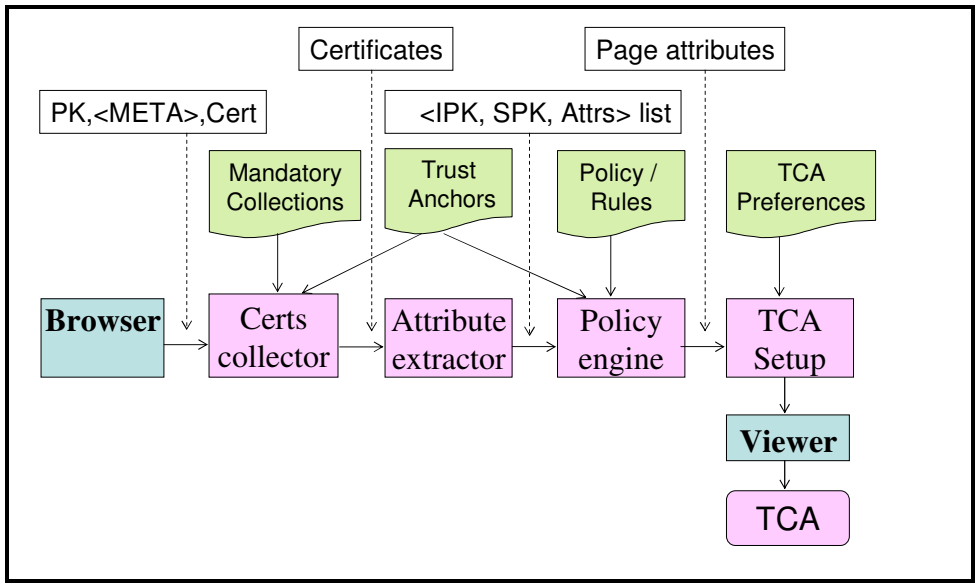


Figure 7: Determining the contents of the Trusted Credentials Area.

The list of *<IPK, SPK, Attrs>* triplets is input to the *Policy Engine module*. This module determines the user-defined properties or roles of the page, based on the set of triplets (which contains the trust information from the collected certificates), together with the user-defined list of trust anchors, and the *policy* or *rules* defined by the user. For example, the policy may indicate that a page has the `my banks` property/role, provided it has a certificate from Citibank™ or from Visa™, with some relevant attributes, and unless there is a `negative` certificate with an `fraud alert` attribute regarding this public key generated by one of the mandatory certificate collections. For more examples of policies and details of a simple policy language and engine, see e.g. [HM*00].

The policy engine outputs a set of *page attributes*; essentially, these attributes identify the logos, seals and other graphical elements for the trusted credentials area (*TCA*). This is input to the *TCA setup module,* which allows the user to customize the TCA; in particular, the user can choose specific images, background, colors, etc.; furthermore, the user may define priorities and rules to reduce the number of images in the TCA and increase readability. To actually display the TCA, the TCA setup module uses the viewer mechanisms included in the base browser (in our case, Mozilla).

4 Efficient Authentication of Response Credentials using CLTLS

As mentioned in Section 2 above, currently only a small fraction of the web pages use SSL (or TLS), since SSL places considerable overhead on both server and client. Specifically, during the entire SSL connection, the server must maintain state identifying the keys and sequence numbers used in the connection. Furthermore and more critical, at the beginning of every connection, SSL performs a handshake process consisting of at least four messages, and computationally-intensive public key operations; the server may save the results of the public key operations for multiple connections with the same client, but this requires storage in the server, and therefore this feature is usually used only for rapid connections. This overhead may be significant, if organizations and corporations will deploy SSL on each of their web pages, to ensure display of their logo and credentials (seals) in trusted area.

There have been several proposals for reducing the SSL handshake overhead, most notably by [BSR02], who proposes client-side caching to reduce communication and possibly server processing load. However, the handshake remains considerable overhead, including additional flows. To allow efficient presentation of credentials and logos in the trusted area, it may be necessary to use an even more aggressive optimization of SSL/TLS that will authenticates only the particular response rather than establish a secure connection. In the next

subsection, we sketch the design of the connection-less transport layer security (CLTLS) protocol. CLTLS is similar to TLS; its advantage is a more efficient validation of web page credentials. We focus only on the use of CLTLS for authentication of credentials and ignore other aspects such as confidentiality (including perfect forward secrecy), identity hiding and denial-of-service protection. In the following subsection we discuss the efficiency of (our use of) CLTLS.

4.1 Simplified description of the Connection-Less Transport Layer Security (CLTLS) protocol

We present the basic flows of the connection-less transport layer security protocol (CLTLS) in Figure 8; our presentation of CLTLS focuses on the features needed for secure authentication of web objects. We will present separately a complete security analysis, as well as design of other important security features (such as confidentiality, including perfect forward secrecy, client/request authentication, identity hiding and protection against denial of service attacks). CLTLS, as we use it, is a simple protocol to authenticate responses in client/server (request/response) setting; for example, a TCA-enabled browser can use it to authenticate web pages and other objects sent to it as responses from web server (over either TCP or UDP). The (full) CLTLS protocol is also an alternative to the Datagram TLS proposal of [MR04], which allows the use of TLS over connectionless channels such as UDP, but without reducing its overhead (in fact, DTLS is slightly more expansive than regular TLS).

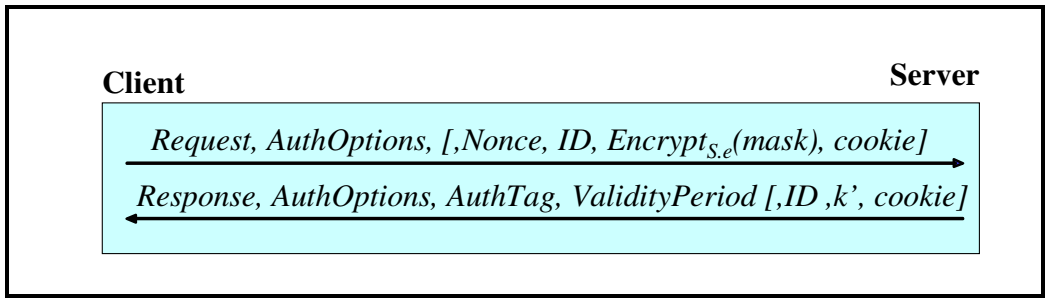


Figure 8: Connection-Less Transport Layer Security Protocol (CLTLS)

In CLTLS, the client sends a request together with the requested authentication options (*AuthOptions*), indicating supported algorithms and certificate/attribute authorities, and other options. The server returns the response together with indication of the authentication options used, an authentication tag *AuthTag* and validity period. As part of *AuthTag,*, the server may optionally send a certificate *S.cert* (containing the server's public signature validation key, *S.v*, and optionally also the server's public encryption key, *S.e*). Also, the server may provide a new shared secret key to the client *k* by sending $k'=k \oplus mask$, where *mask* is a random bit string sent by the client, encrypted using the server's public encryption key *S.e.* In this case, the server will also provide an identifier *ID* for the new shared secret key; and when the client sends a request using the shared secret key, she will include *ID* in the request. The client may also include a random *Nonce* field with the request, which the server should use as part of the input to the authenticator tag *AuthTag,* to detect replay of previous response. Finally, to protect against denial of service attacks attempting to waste server's computational resources by sending many requests, causing the server to perform computationally-intensive sign and/or decrypt private key operations, the server may perform these operations only when the request includes a *cookie* which was sent by the server with a previous response (possibly an empty response sent due to the lack of cookie with the request); *cookie* is computed by the server, e.g. as $cookie=MAC_{ck}(ID)$.

The authenticator tag *AuthTag* may be one or more of the following:

1. Response signed by server: here, $AuthTag=\{Sign_{S.s}(response, request, ValidityPeriod [, Nonce, k]) [, S.cert]\}$, namely a signature by the private signing key of the server (*S.s*) over the response and over a period of time during which the response is valid. This requires that the client knows the server's public key *S.v*, necessary to validate signatures using the secret signing key *S.s*; when *S.v* is not known already (e.g. by prior SSL or

SRAP handshake), the server may provide it by attaching an appropriate public key certificate *S.cert* in the response. When replay of the response is not a threat, there is no need to include the *Nonce* in the signed response (or with the request); in this case, the signature does not depend on the request, and therefore could be cached and pre-computed, for high efficiency when serving static content. The signature includes the request to make sure that the response is related to the specific request.

2. Response authenticated by the server: here, $AuthTag=MAC_k(response, request, ValidityPeriod [, Nonce])$, namely a message authentication code using a key k shared between the server and the client. The key *k* would typically be computed by the server as $k=PRF_{mk}(ID)$, where *mk* is a `master key` kept secretly by the server, *PRF* denotes a pseudo-random function (e.g. implemented using HMAC as in TLS), and *ID* is an identifier for the key *k*, where *ID* includes the time (and/or some other variable fields to prevent replay); both *k* and *ID* were previously selected by the server and provided to the client (e.g. in a previous SRAP or SSL protected connection).
3. Response signed by third-party attribute authorities, identifying credentials and seals associated with the specific contents of the response; here, $AuthTag=\{SignAA.s(response, ValidityPeriod) [, AA.cert]\}$, namely a signature by the private signing key of the attribute authority (*AA.s*) over the response and over a period of time during which the response is valid. This allows third-party seals of quality to specific responses, offerings and products. In particular, this establishes attributes (credentials) of the server or of the contents of a particular page, to prevent spoofing. Furthermore, the attributes may include indicator of advertising content (to prevent spam) or of other properties that may make the browser block the display of the content, e.g. violence or nudity ratings.

Implementation notes:

1. It may be preferable not to send the actual certificates, but instead to send only a URL for the certificates, from which the client can download only needed certificates.
2. Using simple, standard techniques, as in [BSR02], we can ensure interoperability with existing browsers that support SSL/TLS (but not CLTLS).

4.2 Efficiency of CLTLS

We now briefly discuss the efficiency of CLTLS as described above (restricted to authentication of the responses from the server). We first notice that CLTLS is added (`piggybacked`) to the HTTP request/response messages, and therefore does not add any new flows. CLTLS flows that do not contain public key signatures (in responses) and encryptions (in requests), add under 100 bytes The length of requests and responses containing a public key signature (responses) or encryption (requests) is dominated by the length of the public key operations, typically 128 to 256 bytes. For most scenarios, this extra communication is negligible, compared to the length of typical HTTP requests and responses. This dramatically improves upon SSL and TLS, and upon the existing TLS variants such as DTLS [MR04], client-side caching and fast-track TLS [BSR02].

The computational overhead of CLTLS also greatly depends on whether it uses public key operations (encryption for the request, signature for the response). Since the whole purpose of using CLTLS is to authenticate *all* of the organization's web pages (i.e., all pages containing logos or other credentials), we expect that most CLTLS requests will be `repeat requests` to the same server. We expect such `repeat requests` to usually use only a shared key for Message Authentication Code (MAC), whose processing time is comparable to the normal processing time of unprotected messages (e.g. for compressing and encoding for interoperability requirements). In the relatively rare cases of a CLTLS connection without a pre-established shared key, the computational overhead is dominated by the public key operations, and is comparable to that of SSL/TLS.

Finally, we note that CLTLS does not require state (cache) in the web server, similarly to the client-side caching proposal of [BSR02]. Serve caching is a substantial overhead, especially when using multiple server machines for performance (and reliability).

5 Conclusions and Recommendations

As already shown in [FB*97] and in the developer community, currently web users, and in particular naïve users, are vulnerable to different web spoofing attacks; furthermore as shown in [APWG04, L04] and elsewhere, phishing and spoofing attacks are in fact increasingly common. In this paper, we describe browser and protocol extensions that we are designing and implementing, that will help prevent web-spoofing (and phishing) attacks. The main idea is to enhance browsers with a mandatory *Trusted Credentials Area (TCA)*, with a fixed location at the top of every web page, as shown in Figure 6. The most important credential is probably the *Logo* of the organization, used to provide and re-enforce the *brand*; and, when the logo or other credentials are certified by some trusted authority, the logo of that certificate authority.

Our hope is that browser developers will incorporate the trusted credentials area as soon as possible, i.e. make TCA-enabled browsers. We hope to soon make available the source code of our implementation of the TCA (for the Mozilla browser), and we will be happy to cooperate with others on creating high-quality open source code available.

However, to conclude this paper, we present conclusions and recommendations for users and owners of sensitive web sites, such as e-commerce sites, for the period until browser are TCA-enabled. Finally, we conclude by cautioning users and providers, that even when using TCA-enabled browsers, viruses and other malicious software may still be able to create unauthorized transactions, due to operating system vulnerabilities. We recommend that highly sensitive web sites such as e-brokerage consider authorizing transactions using more secure hardware modules (see below).

5.1 Conclusions for Users of Sensitive Web-sites

The focus of this paper was on ensuring security even for naïve web users; however, even expert, cautious users can not be absolutely protected, unless browsers are extended with security measures as we propose or as proposed by [LY03, YS02, YS03]. However, cautious users can increase their security, even before the site incorporates enhanced security measures, by following the following guidelines:

1. Use an TCA-enhanced browser, using its `opportunistic logo identification` mechanism to establish logos for each of your sensitive web-pages. The authors developed and use a simple TCA extension to the Mozilla browser, and plan to make it available for download from their homepages soon (after some final touches).
2. Always contact sensitive web sites by typing their address in the location bar, using a bookmark or following a link from a secure site, preferably protected by SSL/TLS.
3. Never click on links from e-mail messages or from other non-trustworthy sources (such as shady or possibly insecure web sites). These could lead you to a `URL-forwarding` man-in-the-middle attack, which may be hard or impossible to detect, even if you follow guideline 1 above.
4. Be very careful to inspect the location bar and the SSL icon upon entering to sensitive web pages. Preferably, set up your browser to display the details of the certificate upon entering your most sensitive sites (most browsers can do this); this will help you notice the use of SSL and avoid most attacks. Do not trust indications of security and of the use of SSL when they appear as part of the web page, even when this page belongs to trustworthy organizations; see the examples of insecure login pages in Figure 5, by respectable financial institutions and e-commerce sites.
5. If possible, restrict the damages due to spoofing by instructing your financial services to limit online transactions in your account to cover only what you really need. Furthermore, consider using sensitive online services that use additional protection mechanisms beyond SSL, as described below.

5.2 Conclusions for Owners of Sensitive Web-sites

Owners of sensitive web-sites are often financial institutions, with substantial interest in security and ability to influence their consumers and often even software developers. We believe that such entities should seriously consider one of the following solutions:

1. Provide your customers with a browser with security enhancements as described here. We notice that the basic `trusted credentials area` enhancement will suffice for most sites and customers; many software integrators can perform such enhancements to the Mozilla browser easily. We also plan to publish our code for this purpose.
2. Use means of authenticating transactions that are not vulnerable to web spoofing. In particular, `challenge-response` and similar one-time user authentication solutions can be effective against offline spoofing attacks (but may still fail against a determined attacker who is spoofing your web site actively in a `man in the middle` attack). Using SSL client authentication can be even more effective, and avoid the hardware token (but may be more complex and less convenient to the user).
3. Protect, using SSL/TLS, as many of your web pages as is feasible. In particular, be sure that every web page requesting the user to enter sensitive information, is properly protected when it is sent to the user; notice that many respectable companies (probably using respectable web-site designers) were not careful enough and have insecure web pages asking users to enter sensitive information, as shown in Figure 5.

   We also recommend that site owners are careful to educate consumers on the secure web usage guidelines, including these mentioned above, as well as educate them on the structure of domain name and how to identify their corporate domains. This may include restricting corporate domains to only these that end with a clear corporate identity.

5.3 On the secure client requirement

Finally, we notice that even if our recommendations are all implemented, surfers using personal computers are still vulnerable to attacks by malicious software (`malware`) running on their computers, or by attackers who can use the same computer. This is the result of the weak security of existing operating systems, e.g. Microsoft™ issued 51 security advisories during 2003 alone (about one every week!). We therefore recommend, following [PPSW97, H03], to restrict the execution of sensitive transactions to trusted hardware, possibly in the form of a trusted personal device. Such a device can provide a truly high level of confidence in its Trusted Credentials Area, allowing users to identify using user-name and passwords with relatively safety. Furthermore, such a device could support more secure forms of identification and authorization, such as using shared keys and one-time passwords. Finally, a mobile, personal trusted device is also the right mechanism to provide digital signatures with non-repudiation, i.e. allow the server as well as third party (e.g. judge) to validate a digital signature by the customer on submitted transactions and orders; see [H03] for details

Acknowledgements

This work benefited from many fruitful discussions on the cryptography@metzdowd.com mailing list over the last few years, including different ideas and proposals related and similar to ours. We thank the owner and moderator, Perry Metzger, and the many participants. In particular, many thanks to Ian Grigg for his excellent, helpful comments and suggestions.

Thanks to Amos Fiat and Amos Israeli for their encouragement and helpful comments.

This work was support in part by National Science Foundation grant NSF CCR 03-14161 and by Israeli Science Foundation grant ISF 298/03-10.5.

References

[APWG04] Anti-Phishing Working Group, Phishing Attack Trends Report - March 2004, published April 2004, available online at http://www.antiphishing.org/resources.htm.

[BBC03] Virus tries to con PayPal users, BBC News, online at http://news.bbc.co.uk/2/hi/technology/3281307.stm, Wednesday, 19 November, 2003.

[BSR02] Client side caching for TLS. by D. Boneh, Hovav Shacham, and Eric Rescrola.
In proceedings of the Internet Society's 2002 Symposium on Network and Distributed System Security (NDSS), pp. 195—202, 2002.

[CSRI04] The Coordinated Spam Reduction Initiative, Microsoft corporation, February 2004.

[Citi04] Citibank™ corp., Learn About or Report Fraudulent E-mails, at http://www.citibank.com/domain/spoof/report_abuse.htm, April 2004.

[ES00] Carl Ellison and Bruce Schneier**,** Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. Computer Security Journal, v 16, n 1, 2000, pp. 1-7; online at http://www.schneier.com/paper-pki.html.

[FB*97] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web Spoofing: An Internet Con Game. Proceedings of the Twentieth National Information Systems Security Conference, Baltimore, October 1997. Also Technical Report 540–96, Department of Computer Science, Princeton University.

[FS*01] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster, Do's and Don'ts of Client Authentication on the Web, in the Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001.

[G04] Ian Grigg, personal communications, 2004.

[H03] Amir Herzberg, Payments and banking with mobile personal devices. CACM 46(5): 53-58 (2003).

[H04] Amy Harmon, Amazon Glitch Unmasks War Of Reviewers, February 14, 2004.

[He04] Amir Herzberg, Controlling Spam by Secure Internet Content Selection, unpublished manuscript, June 2004.

[HM04] Amir Herzberg, Yosi Mass: Relying Party Credentials Framework. Electronic Commerce Research, Vol. 4, No. 1-2, pp. 23-39, 2004.

[HM*00] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor and Yiftach Ravid: Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. IEEE Symposium on Security and Privacy, Oakland, California, May 2000, pp. 2-14.

[L04] Avivah Litan, Phishing Attack Victims Likely Targets for Identity Theft, Gartner FirstTake, FT-22-8873, Gartner Research, 4 May 2004.

[LY03] Tieyan Li, Wu Yongdong. "Trust on Web Browser: Attack vs. Defense". International Conference on Applied Cryptography and Network Security (ACNS'03). Kunming China. Oct. 16-19, 2003. Springer LNCS.

[LN02] Serge Lefranc and David Naccache, "Cut-&-Paste Attacks with Java". 5th International Conference on Information Security and Cryptology (ICISC 2002), LNCS 2587, pp.1-15, 2003.

[MR04] N. Modadugu, and E. Rescorla. The Design and Implementation of Datagram TLS.
To appear in Proceedings of NDSS 2004.

[Mozilla] http://www.mozilla.org.

[PPSW97] Andreas Pftizmann, Birgit Pfitzmann, Matthias Schunter and Michael Waidner, Trustworthy user devices. In Gunter Muller and Kai Rannenberg, editor, Multilateral Security in Communications, pages 137--156. Addison-Wesley, 1999. Earlier version: Trusting Mobile User Devices and Security Modules, IEEE Computer, 30/2, Feb, 1997, p. 61-68.

[SF9182] Multiple Browser URI Display Obfuscation Weakness, http://www.securityfocus.com/bid/9182/discussion/, Security Focus, December, 2003.

[YS02] Zishuang (Eileen) Ye, Sean Smith: Trusted Paths for Browsers. USENIX Security Symposium 2002, pp. 263-279.

[YYS02] Eileen Zishuang Ye ,Yougu Yuan ,Sean Smith . Web Spoofing Revisited: SSL and Beyond . *Technical Report TR2002-417* February 1, 2002.

[Z95] Phil R. Zimmerman. The Official PGP User's Guide. MIT Press, Boston, 1995.