DIMACS Technical Report 2004-23
May 2004

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

by

Amir Herzberg
Department of Computer Science
Bar-Ilan University
Ramat Gan 52900, Israel
Email: herzbea @. cs.biu.ac.il

Ahmad Gbara
Department of Computer Science
Bar-Ilan University
Ramat Gan 52900, Israel

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. Several papers presented such web spoofing attacks, and 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 non-expert 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.

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 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.

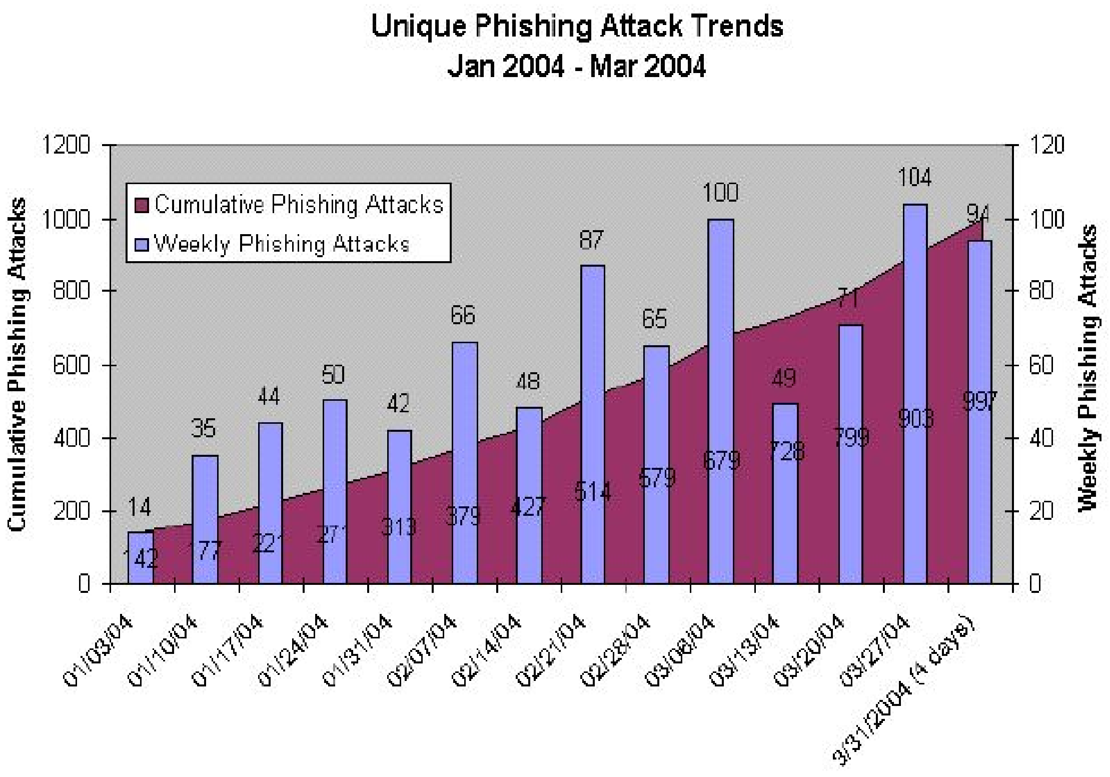


Figure 1: Phishing attack trends (source: [APWG04])

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 SSL/TLS are used by many organizations only for few very 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.

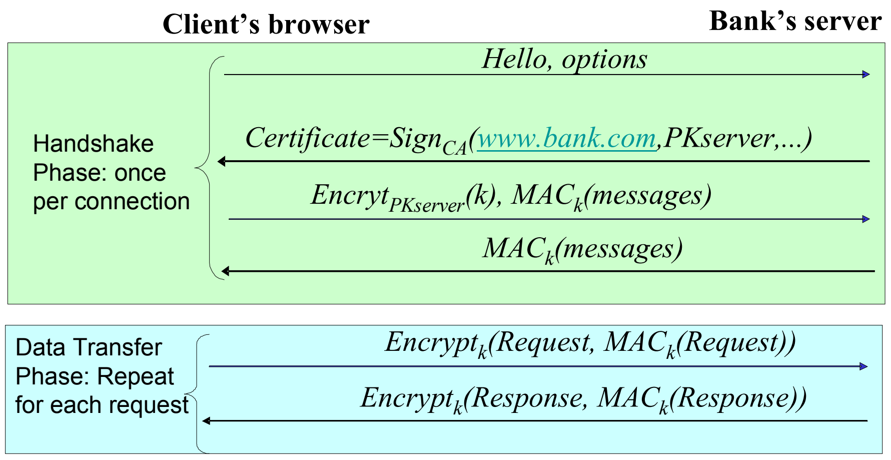

Client's browser
Bank's server
Handshake Phase: once per connection
Hello, options
Certificate=Sign_CA(www.bank.com,PKserver,...)
Encryt_PKserver(k), MAC_k(messages)
MAC_k(messages)
Data Transfer Phase: Repeat for each request
Encrypt_k(Request, MAC_k(Request))
Encrypt_k(Response, MAC_k(Response))

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[1]: 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` *k* 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

[1] 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.

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 described below.

The second, larger vulnerability is that SSL/TLS ultimately depend on the user to validate the authenticity of web sites, by noting relevant status areas in the browser user interface. The most important status areas are the location (address, URL) field 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.

The first challenge for a web spoofing attack is to cause the browser to request the clone site, when the customer is really looking for the victim site. This is easy to achieve for an intercepting adversary, who can intercept user's requests, and send back spoofed contents directly, or redirect them to the spoofed web site. Felten et al. [FB*97] also show similar *URL redirection* attack that begins when the user accesses the attacker's web site, and continues surfing by links from this page. However, this attack also requires the adversary to intercept communication to a web site used by the user, or to control such a site. In practice, attackers usually use an easier method to direct the user to the cloned site: *phishing attack,* using spam e-mail messages.

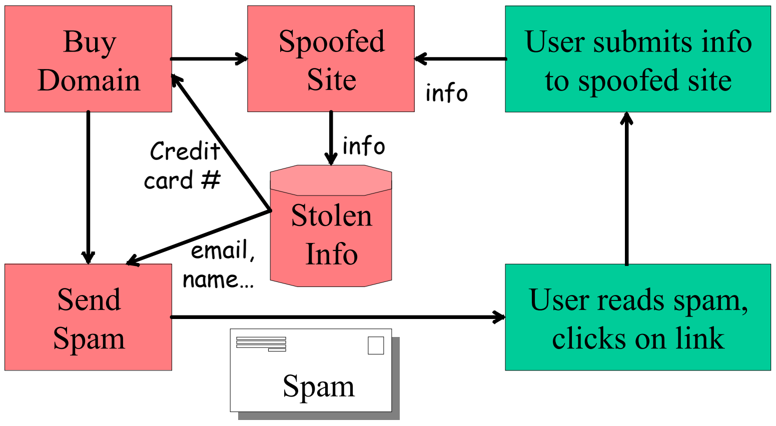


Figure 3: Phishing Attack

In Figure 3 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, convincing them to contact some trusted entity by following a link embedded in the message. 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).

Phishing attacks require only an unallocated domains adversary, i.e. can be launched by essentially any adversary on the Internet; therefore, they are increasingly common, as shown in Figure 1. Most phishing attacks simply 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 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.accounts.com may not)
- Organizations often use multiple domain names, often not incorporating their corporate name, e.g. CitiBank™ uses at least eight domains, e.g. accountonline.com.

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. Many naïve users are likely to believe that the smaller window is also secure and authentic. 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 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 may not suffice for naïve users; in particular, naïve users may not notice the lack of security or the use of incorrect address (URL). Our proposals for secure browser display of credentials incorporate some of these ideas, esp. building on the design criteria of [YS02], to prevent spoofing web pages from creating fake display of credentials.

Notice that phishing attacks exploit spamming, i.e. unsolicited and undesirable e-mail sent to (usually many) recipients. 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 phishing, however this is beyond the scope of this paper.

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 most similar sites 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. Currently, the only way for web sites to present credentials is by including an appropriate picture (often referred to as a seal); 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].

- **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.

- **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.

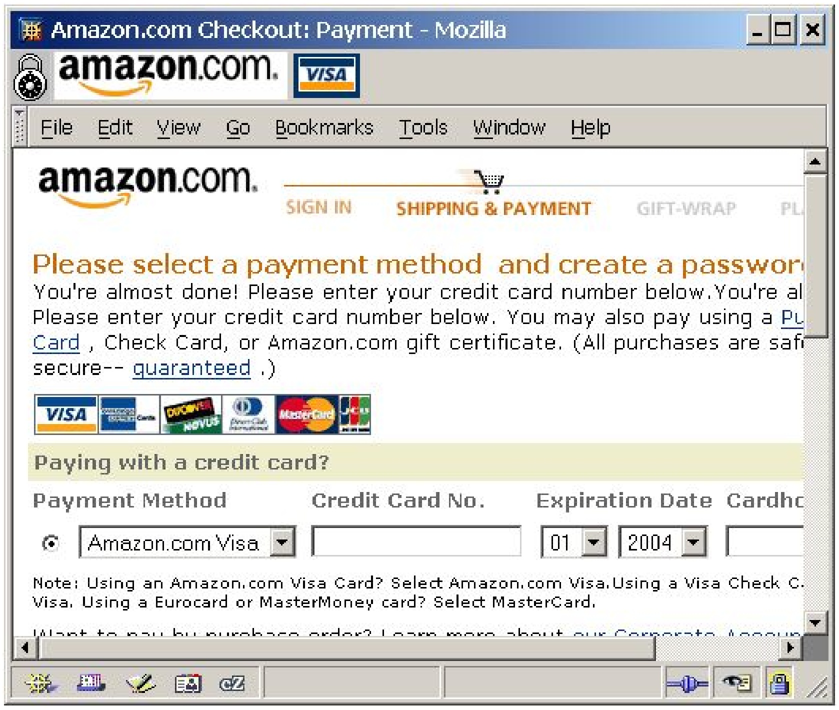
- **Customization:** the visual representation of the different credentials should be customizable by the user. Such customization may make it easier for users to validate credentials, e.g. by allowing users to use the same graphical element for categories of sites, for example for `my financial institutions`. 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 a screen-shot in Figure 4. We refer to a browser supporting a TCA, natively or via an appropriate extension, as *TCA-enabled.*

The main design decision is 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.


Amazon.com Checkout: Payment - Mozilla
amazon.com
VISA
File Edit View Go Bookmarks Tools Window Help
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT-WRAP

Please select a payment method and create a passwor
You're almost done! Please enter your credit card number below.You're al
Please enter your credit card number below. You may also pay using a Pu
Card , Check Card, or Amazon.com gift certificate. (All purchases are saf
secure-- guaranteed .)

Paying with a credit card?

Payment Method	Credit Card No.	Expiration Date		Cardhc
Amazon.com Visa		01	2004	

Note: Using an Amazon.com Visa Card? Select Amazon.com Visa.Using a Visa Check C
Visa. Using a Eurocard or MasterMoney card? Select MasterCard.

Figure 4: Screen-shot of secure site 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). Furthermore, a web site can request that the browser avoid displaying the status area (simply by using the `window.open` JavaScript method); 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.

3.2 Provider Identification

The most basic credential is the identification of the provider of a web page. 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. Graphical identity symbols are recognized as highly effective means for ensuring recognition, including subconscious alert 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.

For SSL protected web pages, the site's public key can be used to identify the logo, since SSL 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. In our current (simple) implementation, the browser extension maintains a *public key / logo table* linking public keys with logos. After SSL completed, our browser extension looks up the public key in the table, retrieve the logos and display them in the trusted area. For example, Figure 4 shows an SSL-protected site, 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 implantation maintains the original SSL indicator unchanged, notice how the logos are more visible. We also added a more visible SSL indicator, in the form of the (larger) lock to the left of the logos, in the trusted credentials area. Furthermore, when visiting an insecure site, a TCA-enabled browser fills the entire trusted credentials area (containing the logos in Figure 4) with a very visible warning message/indicator.

When the client approaches any SSL/TLS protected web site for the first time, the TCA-enabled browser inspects the public key used in the SSL certificate received from the server. If this public is in in the *public key / logo table* then the TCA-enabled browser displays the logo in the TCA.

When no logo is associated with the public key of the site in the table maintained by the browser, the site may 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. In either case, 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*[2] allowing the display of the logo for sites passing SSL handshake with the given public key. Logo certificates are issued by *Logo Certificate Authority (LCA)* entities. In a logo certificate, the logo certificate authority establishes the connection between the public key and the given logo. 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 be fairly short, and well understood by users; in fact, considering that there are international recognized bodies for registering trademarks (including logos), it seems quite possible that browsers will simply contain the public keys of the most important of these. 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 could also display the logo of the LCA; for example, in Figure 4, VISA™ may be a LCA, who have signed the Amazon™ logo (in the figure the logos are displayed simply side by side, but one could also design display conventions to illustrate the fact that the Amazon™ logo was validated by Visa™).

To facilitate gradual adoption of the Trusted Credentials Area (TCA) and of logo certificates, we propose that TCA-enabled browsers would also allow *opportunistic logo identification*. Namely, when a TCA-enabled browser detects a proposed logo, but without an appropriate logo certificate or a logo certificate from an unknown LCA, it may present a dialog to the user, displaying the logo, and the following options:

- Approve the presentation of this logo whenever accessing this site

[2] Using X.509 terminology, this could be either a public key certificate or an attribute certificate.

- Approve the presentation of this logo just for this session
- Present an alternate logo or icon whenever accessing this site, 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).
- Do not use any logo or icon for this site; present the URL (or other text) in the Trusted Credentials Area.

Even for SSL-protected sites that do not offer a logo, e.g. existing sites, a TCA-enabled browser could still present a dialog box to allow the user to define a logo for this site. The dialog can contain some identification information taken from the certificate, to help the user select the logo or text. Such *opportunistic identification* provides significant security advantage to early users of the spoofing-prevention browser extension, even when approaching existing, `legacy` web sites (protected by SSL/TLS).

3.3 Establishing 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, 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.

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 `reli-

able` 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 Process and architecture for determining the contents of the Trusted Credentials Area

We present the process for collecting (positive and negative) credentials in Figure 5; 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].

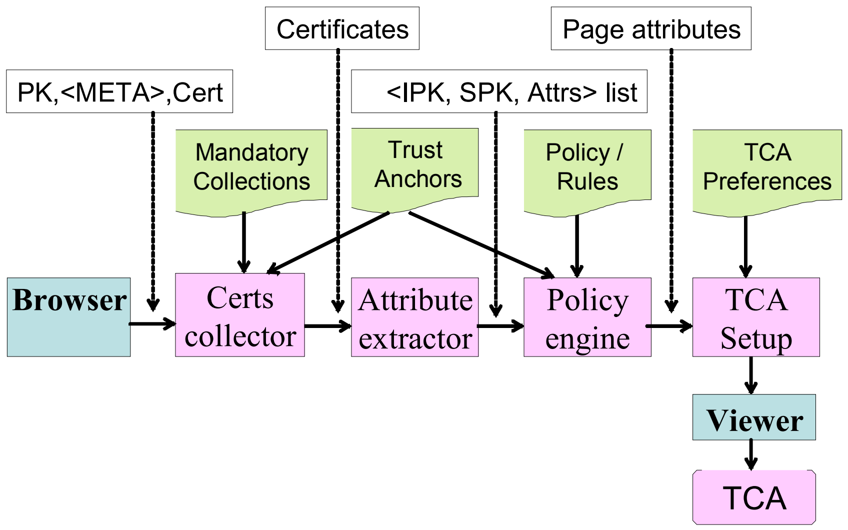


Figure 5: Determining the contents of the Trusted Credentials Area.

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 a variant of the TLS protocol, which allows much more efficient validation of the credentials of web sites; 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 6; 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).

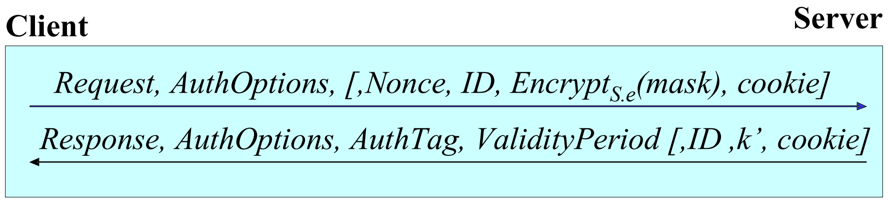

Client
Server
Request, AuthOptions, [,Nonce, ID, Encrypt_{S.e}(mask), cookie]
Response, AuthOptions, AuthTag, ValidityPeriod [,ID ,k', cookie]

Figure 6: 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.
3. 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] 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 4.

Our hope is that browser developers will incorporate the trusted credentials area as soon as possible, i.e. make TCA-enabled browsers. 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).

5.0.1 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. Always contact sensitive web sites by typing their address in the location bar, using a bookmark or following a link from a secure site.
2. 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.
3. 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.
4. 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.

5.0.2 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:

- 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.

- 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).

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.

5.0.3 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

6 Acknowledgements

This work was support in part by National Science Foundation grant NSF CCR 03-14161.

7 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.

[Citi] 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.

[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.

[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.